

INFORMIX[®]-Universal Server

Administrator's Guide

Version 9.1
March 1997
Part No. 000-3851

Published by INFORMIX® Press

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

Copyright © 1981-1997 by Informix Software, Inc. or their 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:

INFORMIX®; INFORMIX®-OnLine Dynamic Server™; Illustra™; DataBlade®

The following are worldwide trademarks of the indicated owners or their subsidiaries, registered in the United States of America as indicated by “®,” and in numerous other countries worldwide:

Adobe Systems Incorporated: PostScript®
Novell, Inc.: Novell®; NetWare®, IPX/SPX™
UNIX System Laboratories: OPEN LOOK®
X/Open Company Ltd.: UNIX®, X/Open®

All other marks or symbols are registered trademarks or trademarks of their respective owners.

Documentation Team: Bob Berry, Diana Chase, Beth Delson, John Francis, Patrice O’Neill, Judith Sherwood

To the extent that this software allows the user to store, display, and otherwise manipulate various forms of data, including, without limitation, multimedia content such as photographs, movies, music and other binary large objects (blobs), use of any single blob may potentially infringe upon numerous different third-party intellectual and/or proprietary rights. It is the user's responsibility to avoid infringements of any such third-party rights.

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 of Defense use, with rights as restricted by vendor's standard license, unless superseded by negotiated vendor license as prescribed in DFAR 227.7202.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-Universal Server?

Chapter 1 What Is INFORMIX-Universal Server?

Chapter 2 Overview of Administration

Section II Configuration

Chapter 3 Installing and Configuring

Chapter 4 Client/Server Communications

Chapter 5 What Is Multiple Residency?

Chapter 6 Using Multiple Residency

Section III Modes and Initialization

Chapter 7 What Are the Operating Modes?

Chapter 8 Managing Modes

Chapter 9 What Is Initialization?

Section IV Disk, Memory, and Process Management

Chapter 10 What Is the Universal Server Architecture?

Chapter 11 Managing Virtual Processors

Chapter 12 Shared Memory

Chapter 13 Managing Shared Memory

Chapter 14 Where Is Data Stored?

Chapter 15 Managing Disk Space

Chapter 16 What Is Fragmentation?

Chapter 17 Managing Fragmentation

Chapter 18 What Is PDQ?

Chapter 19 Managing PDQ and Decision Support

Section V Logging and Log Administration

Chapter 20 What Is Logging?

Chapter 21 Managing Database-Logging Status

Chapter 22 What Is the Logical Log?

Chapter 23 Managing Logical-Log Files

Chapter 24 What Is Physical Logging?

Chapter 25 Managing the Physical Log

Chapter 26 What Is Fast Recovery?

Section VI Fault Tolerance

Chapter 27 What Is Mirroring?

Chapter 28 Using Mirroring

Chapter 29 What Is Data Replication?

Chapter 30 Using Data Replication

Chapter 31 What Is Consistency Checking?

Chapter 32 Situations to Avoid

Section VII Monitoring INFORMIX-Universal Server

Chapter 33 Monitoring the Database Server

Section VIII Distributed Data

Chapter 34 Multiphase Commit Protocols

Chapter 35 Recovering Manually from Failed Two-Phase Commit

Section IX Reference

Chapter 36 ON-Monitor

Chapter 37 Configuration Parameters

Chapter 38 The sysmaster Database

Chapter 39 Utilities

Chapter 40 Message-Log Messages

Chapter 41 Interpreting Logical-Log Records

Chapter 42 Disk Structures and Storage

Table of Contents

Introduction

About This Manual	3
Organization of This Manual	3
Types of Users	4
Software Dependencies	4
Assumptions About Your Locale.	5
Demonstration Database	5
Major Features	6
Documentation Conventions	7
Typographical Conventions	7
Icon Conventions	8
Command-Line Conventions	9
Additional Documentation	12
On-Line Manuals	12
Printed Manuals	12
Error Message Files	13
Documentation Notes, Release Notes, Machine Notes	13
Compliance with Industry Standards	14
Informix Welcomes Your Comments	14

Section I What Is INFORMIX-Universal Server?

Chapter 1 What Is INFORMIX-Universal Server?

What Is Universal Server?	1-3
Complex and User-Defined Data Types	1-6
Support for Large Objects	1-7
Support for External Spaces	1-8
Inheritance	1-8
User-Defined Routines and DataBlade Modules	1-8
User-Defined Access Methods	1-9

Universal Server Features	1-9
Client/Server Architecture	1-10
Scalability	1-11
High Performance	1-11
Fault Tolerance and High Availability	1-13
Distributed Data Queries	1-16
Database Server Security	1-16
Who Uses Universal Server?	1-17
Database Users	1-17
Database Administrators	1-17
Universal Server Administrators	1-18
Application Developers	1-18
Features Beyond the Scope of Universal Server	1-18
No Bad-Sector Mapping	1-19
No Scanning or Compression of Simple Large Objects	1-19

Chapter 2 **Overview of Administration**

Initial Tasks	2-3
Routine Tasks	2-4
Changing Modes	2-4
Backing Up Data and Backing Up Logical-Log Files	2-4
Monitoring Activity	2-5
Checking for Consistency	2-5
Configuration Tasks	2-5
Managing Disk Space	2-5
Managing Database-Logging Status	2-6
Managing Extensions	2-6
Logical-Log Administration	2-6
Physical-Log Administration	2-7
Using Auditing	2-7
Using Mirroring	2-7
Using Data Replication	2-7
Managing Shared Memory	2-8
Managing Virtual Processors	2-8
Managing Parallel Database Query	2-9

Section II Configuration

Chapter 3 Installing and Configuring

Planning	3-4
Consider Your Priorities	3-4
Consider Your Resources	3-5
Administering	3-6
Installing	3-7
Installing When No Other Informix Products Are Present	3-7
Installing When Other Informix Products Are Present	3-8
Upgrading a Version of INFORMIX-OnLine	3-8
Creating a Private Installation	3-9
Running Multiple Universal Server Database Servers.	3-10
Configuration Overview	3-10
Configuration Files	3-11
Environment Variables That Universal Server Uses	3-11
Configuring Universal Server	3-13
Set Environment Variables	3-13
Prepare the ONCONFIG Configuration File	3-14
Overview of Configuration Parameters	3-15
Allocate Disk Space	3-26
Prepare the Connectivity File	3-27
Prepare the ON-Archive Configuration File	3-27
Prepare for Global Language Support	3-27
Evaluate UNIX Kernel Parameters	3-27
Start the Database Server and Initialize Disk Space	3-28
Create Dbspaces, Sbspaces, and Blobspaces	3-28
Configure Extspaces	3-29
Perform Administrative Tasks.	3-29

Chapter 4 Client/Server Communications

What Is Client/Server Architecture?	4-3
What Is a Network Protocol?	4-4
Which Network Protocols Does the Database Server Support?	4-4
Which Network Protocol Should You Use?	4-5
What Is a Network Programming Interface?	4-5
Which Network Programming Interfaces Does the Database Server Support?	4-6
What Interface/Protocol Combinations Are Available on Your Platform?.	4-6

What Are Communications Support Services?	4-6
What is a Communications Support Module?	4-7
How to Configure the DCE-GSS Communications Support Module	4-8
What Is a Connection?	4-10
How Does a Client Application Connect to a Database Server? .	4-10
What Connections Does the Database Server Support?	4-10
Local Connections	4-11
Shared-Memory Connections	4-11
Stream-Pipe Connections	4-12
Network Connections	4-13
PC Connections	4-16
Connectivity Files	4-18
Network-Configuration Files.	4-19
Network-Security Files.	4-21
The \$INFORMIXDIR/etc/concsm.cfg file	4-23
The \$INFORMIXDIR/etc/sqlhosts File	4-24
ONCONFIG Parameters for Connectivity	4-44
The DBSERVERNAME Configuration Parameter.	4-44
The DBSERVERALIASES Configuration Parameter	4-45
Environment Variables for Network Connections	4-46
Examples of Client/Server Configurations	4-46
Using a Shared-Memory Connection	4-47
Using a Local Loopback Connection	4-48
Using a Network Connection	4-49
Using Multiple Connection Types	4-50
Accessing Multiple Database Servers	4-52
Using the Relay Module	4-53
Using 5.x INFORMIX-STAR or 5.x INFORMIX-NET	4-57
Using a Post-6.0 Client Application with a 5.x Database Server .	4-58

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	5-4
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	6-4
Set Your ONCONFIG Environment Variable	6-5

Edit the New Configuration File	6-5
Add Connection Information	6-6
Update the \$INFORMIXDIR/etc/sqlhosts File	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

Section III Modes and Initialization

Chapter 7 What Are the Operating Modes?

Off-Line Mode	7-3
Quiescent Mode	7-4
On-Line Mode	7-4
Read-Only Mode	7-4
Recovery Mode	7-4
Shutdown Mode	7-5

Chapter 8 Managing Modes

Users Permitted to Change Modes	8-3
From Off-Line to Quiescent	8-3
How to Perform This Change Using ON-Monitor	8-4
How to Perform This Change Using oninit	8-4
From Off-Line to On-Line	8-4
How to Perform This Change Using oninit	8-4
From Quiescent to On-Line	8-5
How to Perform This Change Using ON-Monitor	8-5
How to Perform This Change Using onmode	8-5
Gracefully from On-Line to Quiescent	8-5
How to Perform This Change Using ON-Monitor	8-6
How to Perform This Change Using onmode	8-6
Immediately from On-Line to Quiescent	8-6
How to Perform This Change Using ON-Monitor	8-7
How to Perform This Change Using onmode	8-7
From Any Mode Immediately to Off-Line	8-7
How to Perform This Change Using ON-Monitor	8-8
How to Perform This Change Using onmode	8-8

Chapter 9

What Is Initialization?

Types of Initialization	9-3
Initialization Commands	9-4
Initialization Steps	9-4
Process Configuration File	9-5
Create Shared-Memory Portions	9-6
Initialize Shared-Memory Structures	9-7
Initialize Disk Space	9-8
Start All Required Virtual Processors	9-8
Make Necessary Conversions	9-8
Initiate Fast Recovery	9-8
Initiate a Checkpoint	9-9
Document Configuration Changes	9-9
Create the oncfg_servername.servernum File	9-9
Drop Temporary Tblspaces	9-9
Set Forced Residency If Specified	9-10
Return Control to User	9-10
Prepare SMI Tables	9-10
After Initialization	9-11

Section IV Disk, Memory, and Process Management

Chapter 10

What Is the Universal Server Architecture?

What Is a Virtual Processor?	10-5
What Is a Thread?	10-6
Types of Virtual Processors	10-7
Advantages of Virtual Processors	10-8
How Virtual Processors Service Threads	10-12
Control Structures	10-12
Context Switching	10-13
Stacks	10-15
Queues	10-16
Mutexes	10-18
Virtual-Processor Classes	10-19
CPU Virtual Processors	10-19
User-Defined Classes of Virtual Processors	10-24
Disk I/O Virtual Processors	10-26
Network Virtual Processors	10-31
Optical Virtual Processor	10-39
Audit Virtual Processor	10-39
Miscellaneous Virtual Processor	10-39

Chapter 11 Managing Virtual Processors

Setting Virtual-Processor Configuration Parameters	11-3
Setting Virtual-Processor Configuration Parameters with ON-Monitor.	11-4
Setting Virtual-Processor Configuration Parameters with a Text Editor	11-6
Starting and Stopping Virtual Processors	11-7
Adding Virtual Processors in On-Line Mode.	11-8
Dropping CPU or User-Defined Virtual Processors in On-Line Mode	11-10

Chapter 12 Shared Memory

What Is Shared Memory?	12-5
How the Database Server Uses Shared Memory	12-6
How the Database Server Allocates Shared Memory	12-7
How Much Shared Memory Does the Database Server Need?.	12-9
What Action Should You Take If SHMTOTAL Is Exceeded?	12-10
What Processes Attach to Database Server Shared Memory?.	12-11
How a Client Attaches to the Communications Portion	12-11
How Utilities Attach to Shared Memory	12-12
How Virtual Processors Attach to Shared Memory.	12-12
The Resident Portion of Database Server Shared Memory	12-17
Shared-Memory Header.	12-19
Shared-Memory Internal Tables	12-19
Shared-Memory Buffer Pool	12-24
The Virtual Portion of Database Server Shared Memory	12-28
How the Database Server Manages the Virtual Portion of Shared Memory	12-28
The Virtual Portion of Shared Memory.	12-29
The Communications Portion of Shared Memory	12-33
Virtual Extension Portion of Shared Memory	12-34
Concurrency Control	12-34
Shared-Memory Mutexes	12-34
Shared-Memory Buffer Locks	12-35
How Database Server Threads Access Shared Buffers	12-36
LRU Queues	12-36
Configuring the Database Server to Read Ahead	12-41
How a Database Server Thread Accesses a Buffer Page	12-42
How the Database Server Flushes Data to Disk	12-45
Flushing the Physical-Log Buffer.	12-46
How the Database Server Synchronizes Buffer Flushing.	12-49

How Write Types Describe Flushing Activity	12-50
Flushing the Logical-Log Buffer.	12-52
How the Database Server Achieves Data Consistency	12-55
Critical Sections	12-55
Checkpoints	12-56
Time Stamps	12-59
Buffering Large-Object Data	12-61
Writing Simple-Large-Object Data	12-61
Accessing Smart Large Objects	12-64

Chapter 13 Managing Shared Memory

Setting Shared-Memory Configuration Parameters	13-3
UNIX Kernel Configuration Parameters	13-4
Shared-Memory Configuration Parameters.	13-6
Reinitializing Shared Memory	13-14
Turning Residency On or Off for Resident Shared Memory	13-14
Turning Residency On or Off in On-Line Mode	13-14
Turning Residency On or Off for the Next Time You Reinitialize Shared Memory	13-15
Adding a Segment to the Virtual Portion of Shared Memory	13-15
Forcing a Checkpoint	13-16

Chapter 14 Where Is Data Stored?

Overview of Data Storage	14-3
What Are the Physical Units of Storage?	14-5
What Is a Chunk?	14-5
What Is a Page?	14-10
What are Blobpages and Sbpages?	14-11
What Is an Extent?	14-13
What Are the Logical Units of Storage?	14-16
What Is a Dbospace?	14-16
What Is a Blobspace?	14-23
What Is an Sospace?	14-24
What Is a Extspace?	14-25
What Is a Database?	14-25
What Is a Table?	14-26
What Is a Temporary Table?	14-28
What Is a Tblspace?	14-31
How Much Disk Space Do You Need to Store Your Data?	14-33
Calculate the Size of the Root Dbospace	14-33
Estimate Space That Databases Require	14-36

Disk-Layout Guidelines	14-36
General Dbspace/Chunk Guidelines	14-37
Table-Location Guidelines	14-39
Sample Disk Layouts	14-43
What Is a Logical-Volume Manager?	14-49

Chapter 15 **Managing Disk Space**

Allocating Disk Space	15-4
Allocating Cooked File Space	15-5
Allocating Raw Disk Space	15-5
Initializing Disk Space	15-8
Initializing Disk Space with ON-Monitor	15-8
Initializing Disk Space with oninit	15-9
Creating a Dbspace	15-9
Creating a Dbspace with ON-Monitor	15-10
Creating a Dbspace with onspaces	15-11
Adding a Chunk to a Dbspace	15-12
Adding a Chunk	15-12
Creating a Blobospace	15-14
Determining Universal Server Page Size	15-15
Creating a Blobospace with ON-Monitor	15-15
Creating a Blobospace with onspaces.	15-16
Adding a Chunk to a Blobospace	15-16
Creating a Sbspace	15-16
Sizing Sbspace Metadata	15-17
Adding a Chunk to a Sbspace	15-18
Dropping a Chunk from a Dbspace with onspaces	15-18
Dropping a Chunk from a Blobospace or Sbspace	15-19
Dropping a Dbspace, Blobospace, or Sbspace	15-19
Dropping a Dbspace or Blobospace with ON-Monitor	15-20
Dropping a Dbspace, Blobospace, or Sbspace with onspaces.	15-20
Optimizing Blobospace Blobpage Size	15-21
Determining Blobospace Storage Efficiency	15-21
Blobospace Storage Statistics.	15-22
Determining Blobpage Fullness with oncheck -pB	15-22
Managing Extspaces	15-25
Creating a Extspace	15-25
Dropping a Extspace	15-26

Chapter 16	What Is Fragmentation?	
	What Is Fragmentation?	16-3
	Why Use Fragmentation?	16-5
	Whose Responsibility Is Fragmentation?.	16-6
	Fragmentation and Logging	16-7
	Can You Fragment a Temporary Table?	16-7
	Distribution Schemes for Table Fragmentation	16-7
	Round-Robin Distribution Scheme.	16-9
	Expression-Based Distribution Schemes	16-10
	When Can the Database Server Eliminate Fragments from a Search?	16-12
	Fragmentation of Table Indexes	16-17
	Attached Index	16-17
	Detached Index	16-18
	Restrictions on Indexes for Fragmented Tables	16-18
	Internal Structure of Fragmented Tables	16-19
	Disk Allocation	16-19
	Rowids	16-20
	Skipping Inaccessible Fragments	16-21
	Effect of the Data-Skip Feature on Transactions	16-22
	When to Use Data Skip.	16-22
	Formulating a Fragmentation Strategy	16-23
	Deciding How To Fragment a Table	16-25
	Which Distribution Scheme Should You Use?	16-29
	How Many Fragments Should You Use?.	16-32
Chapter 17	Managing Fragmentation	
	Fragmentation Tasks with SQL Statements	17-3
	Skipping Fragments	17-4
	Using the DATASKIP Configuration Parameter	17-4
	Using the SET DATASKIP Statement	17-5
	Monitoring Fragmentation Use	17-6
Chapter 18	What Is PDQ?	
	What Is PDQ?	18-4
	What Types of Applications Use Universal Server?	18-6
	OLTP Applications	18-6
	Decision-Support Applications	18-7
	When Should You Use PDQ?	18-8
	Processing OLTP Queries	18-9
	Processing Decision-Support Queries.	18-10

How Does the Database Server Allocate Resources with PDQ?	18-11
Parameters Used for Controlling PDQ	18-11
How Does the Database Server Use PDQ?	18-14
SQL Operations That Take Advantage of PDQ	18-14
SQL Operations That Do Not Use PDQ	18-17
Update Statistics and PDQ	18-18
Stored Procedures, Triggers, and PDQ	18-18
Correlated and Uncorrelated Subqueries and PDQ	18-19
Remote Tables and PDQ	18-19
How Does the Database Server Manage PDQ Queries?	18-20
How Does PDQ Interact with Parallel Sorts?	18-21
Sorting When PDQ Priority Is Zero	18-21
Sorting When PDQ Priority Is Greater Than Zero	18-21

Chapter 19 **Managing PDQ and Decision Support**

Controlling the Use of Resources	19-3
Limiting the Priority of PDQ Queries	19-4
Adjusting the Amount of Memory	19-7
Limiting the Number of Concurrent Scans	19-8
Limiting the Maximum Number of Queries	19-9
Managing Applications	19-9
Using SET EXPLAIN	19-9
Using OPTCOMPIND	19-10
Using SET PDQPRIORITY	19-10
End-User Control of Resources	19-11
Database Administrator Control of Resources	19-11
Using the Memory Grant Manager	19-12

Section V **Logging and Log Administration**

Chapter 20 **What Is Logging?**

Which Database Server Processes Require Logging?	20-3
What Database Server Activity Is Logged?	20-5
Activity That Is Always Logged	20-6
Activity Logged for Databases with Transaction Logging	20-7
Are Simple Large Objects Logged?	20-7
Are Smart Large Objects Logged?	20-8
What Is Transaction Logging?	20-9
The Database-Logging Status	20-10
When to Use Transaction Logging	20-12

When to Buffer Transaction Logging	20-12
Who Can Set or Change Logging Status	20-13

Chapter 21 Managing Database-Logging Status

About Changing Logging Status	21-3
Modifying Database-Logging Status with ON-Archive	21-5
Turning On Transaction Logging with ON-Archive	21-5
Canceling a Logging Operation with ON-Archive	21-6
Ending Logging with ON-Archive	21-6
Changing Buffering Status with ON-Archive	21-6
Making a Database ANSI Compliant with ON-Archive	21-7
Modifying Database-Logging Status with ontape	21-7
Turning On Transaction Logging with ontape	21-7
Ending Logging with ontape	21-8
Changing Buffering Status with ontape	21-8
Making a Database ANSI Compliant with ontape	21-8
Modifying Database Logging Status with ON-Monitor	21-9

Chapter 22 What Is the Logical Log?

What Is the Logical Log?	22-3
What Is a Logical-Log File?	22-4
How Big Should the Logical Log Be?	22-5
Performance Considerations	22-5
Long-Transaction Consideration	22-6
Smart-Large-Object User-Data Logging Consideration	22-6
Smart-Large-Object Long-Transaction Consideration	22-7
Logical-Log Size Guidelines	22-7
Determining the Size of the Logical Log	22-8
Preserving Log Space for ON-Archive Tasks	22-8
Enabling the Logs-Full High-Water Mark	22-10
Emergency Log Backup	22-10
What Should Be the Size and Number of Logical-Log Files?	22-12
Where Should Logical-Log Files Be Located?	22-13
How Are Logical-Log Files Identified?	22-13
What Are the Status Flags of Logical-Log Files?	22-14
Point-In-Time Recovery	22-16
Why Do Logical-Log Files Need to Be Backed Up?	22-16
When Are Logical-Log Files Freed?	22-17
When Does Universal Server Attempt to Free a Log File?	22-17
What Happens If the Next Logical-Log File Is Not Free?	22-17
Avoiding Long Transactions	22-19

What Are the Logical-Log Administration Tasks Required for Blobspaces?	22-23
Switching Logical-Log Files to Activate Blobspaces	22-23
Switching Logical-Log Files to Activate New Blobspace Chunks	22-23
Backing Up Logical-Log Files to Free Blobpages	22-24
What Is the Logging Process?	22-26
Dbpace Logging	22-26
Blobspace Logging.	22-28

Chapter 23 Managing Logical-Log Files

Adding a Logical-Log File	23-4
Using ON-Monitor to Add a Log File	23-4
Using onparams to Add a Log File	23-5
Adding a Log File with a New Size	23-5
Dropping a Logical-Log File	23-6
Using ON-Monitor to Drop a Logical-Log File	23-7
Using onparams to Drop a Logical-Log File	23-7
Moving a Logical-Log File to Another Dbpace	23-8
An Example of Moving Logical-Log Files	23-8
Changing the Size of Logical-Log Files	23-9
Changing Logical-Log Configuration Parameters	23-10
Changing LOGSIZE or LOGFILES	23-10
Changing LOGSMAX, LTXHWM, or LTXEHWL	23-12
Freeing a Logical-Log File	23-13
Freeing a Log File with Status A	23-13
Freeing a Log File with Status U	23-14
Freeing a Log File with Status U-B	23-14
Freeing a Log File with Status U-C or U-C-L	23-14
Freeing a Log File with Status U-B-L	23-15
Switching to the Next Logical-Log File	23-15

Chapter 24 What Is Physical Logging?

What Is Physical Logging?	24-3
What Is the Purpose of Physical Logging?	24-4
What Database Server Activity Is Physically Logged?	24-4
How Big Should the Physical Log Be?	24-5
Where Is the Physical Log Located?	24-8
Details of Physical Logging	24-9
Page Is Read into the Shared-Memory Buffer Pool	24-9
A Copy of the Page Buffer Is Stored in the Physical-Log Buffer	24-10
Change Is Reflected in the Data Buffer	24-10
Physical-Log Buffer Is Flushed to the Physical Log	24-10

Page Buffer Is Flushed	24-10
When a Checkpoint Occurs	24-11
How the Physical Log Is Emptied	24-11

Chapter 25 Managing the Physical Log

Changing the Physical-Log Location and Size	25-3
Why Change Physical-Log Location and Size?.	25-3
Before You Make the Changes	25-4
Using ON-Monitor to Change Physical-Log Location or Size.	25-4
Using a Text Editor to Change Physical-Log Location and Size	25-5
Using onparams to Change Physical-Log Location or Size	25-5

Chapter 26 What Is Fast Recovery?

What Is Fast Recovery?	26-3
When Is Fast Recovery Needed?	26-4
When Does the Database Server Initiate Fast Recovery?	26-4
Fast Recovery and Buffered Logging	26-4
Fast Recovery and No Logging	26-5
Details of Fast Recovery	26-5
Returning to the Last-Checkpoint State	26-6
Finding the Checkpoint Record in the Logical Log	26-6
Rolling Forward Logical-Log Records	26-7
Rolling Back Incomplete Transactions	26-7

Section VI Fault Tolerance

Chapter 27 What Is Mirroring?

What Is Mirroring?	27-4
What Are the Benefits of Mirroring?	27-4
What Are the Costs of Mirroring?	27-5
What Happens If You Do Not Mirror?	27-5
What Should You Mirror?.	27-5
What Mirroring Alternatives Exist?	27-6
The Mirroring Process	27-7
What Happens When You Create a Mirrored Chunk?	27-7
What Are Mirror Status Flags?	27-8
What Is Recovery?	27-8
What Happens During Processing?	27-9
What Happens If You Stop Mirroring?	27-11
What Is the Structure of a Mirrored Chunk?	27-11

Chapter 28

Using Mirroring

Steps Required for Mirroring Data	28-3
Enabling Mirroring	28-4
Allocating Disk Space for Mirrored Data	28-5
Starting Mirroring	28-6
Mirroring the Root Dbspace During Initialization	28-7
Starting Mirroring for Unmirrored Dbspaces	28-8
Starting Mirroring for New Dbspaces	28-8
Adding Mirrored Chunks	28-9
Changing the Mirror Status	28-10
Taking Down a Mirrored Chunk	28-10
Recovering a Mirrored Chunk.	28-11
Relinking a Chunk to a Device After a Disk Failure	28-12
Ending Mirroring	28-12
Ending Mirroring with ON-Monitor	28-13
Ending Mirroring with onspaces	28-13

Chapter 29

What Is Data Replication?

What Is Data Replication?	29-3
What Is High-Availability Data Replication?	29-4
How Does Data Replication Work?	29-8
How Is the Data Initially Replicated?	29-8
Reproducing Updates to the Primary Database Server	29-9
What Threads Handle Data Replication?	29-13
Checkpoints Between Database Servers	29-14
How Is Data Synchronization Tracked?	29-14
Data-Replication Failures	29-15
What Are Data-Replication Failures?	29-15
How Are Data-Replication Failures Detected?	29-16
What Happens When a Data-Replication Failure Is Detected?	29-16
Considerations After Data-Replication Failure	29-17
Redirection and Connectivity for Data-Replication Clients	29-22
Designing Clients for Redirection	29-22
Automatic Redirection with DBPATH	29-23
Administrator-Controlled Redirection with the sqlhosts File	29-24
User-Controlled Redirection with INFORMIXSERVER	29-28
Handling Redirection Within an Application	29-29
Comparison of Different Redirection Mechanisms	29-32
Designing Data-Replication Clients	29-33
Setting Lock Mode to Wait for Access to Primary Database Server	29-33
Designing Clients to Use the Secondary Database Server	29-34

Chapter 30

Using Data Replication

Planning for Data Replication	30-3
Configuring Data Replication	30-4
Meeting Hardware and Operating-System Requirements	30-4
Meeting Database and Data Requirements	30-5
Meeting Database Server Configuration Requirements.	30-6
Configuring Data-Replication Connectivity.	30-8
Starting Data Replication for the First Time	30-10
Performing Basic Universal Server Administration Tasks	30-14
Changing Database Server Configuration Parameters	30-14
Dbospace and Logical-Log File Backups	30-15
Changing the Logging Status of Databases	30-16
Adding and Dropping Chunks, Dbospaces, and Blobspaces	30-16
Using and Changing Mirroring of Chunks	30-16
Managing the Physical Log	30-17
Managing the Logical Log	30-18
Managing Virtual Processors.	30-18
Managing Shared Memory	30-18
Changing the Database Server Mode	30-19
Changing the Database Server Type	30-20
Changing the Database Server Type of the Primary Database Server	30-21
Changing the Database Server Type of the Secondary Database Server	30-21
Restoring Data If Media Failure Occurs	30-22
Restoring After Media Failure on the Primary Database Server	30-22
Restoring After Media Failure on the Secondary Database Server	30-24
Restarting Data Replication After a Failure	30-26
Restarting After Critical Data Is Damaged	30-26
Restarting If Critical Data Is Not Damaged	30-28

Chapter 31

What Is Consistency Checking?

Performing Periodic Consistency Checking	31-3
Verify Consistency	31-4
Monitor for Data Inconsistency	31-6
Retain Consistent Level-0 Dbospace.	31-8
Dealing with Corruption	31-8
Symptoms of Corruption	31-9
Run oncheck First	31-9
I/O Errors on a Chunk	31-10
Collecting Diagnostic Information	31-10

Chapter 32 Situations to Avoid

Situations to Avoid in Administering the Database Server	32-3

Section VII Monitoring INFORMIX-Universal Server

Chapter 33 Monitoring the Database Server

Sources of Information for Monitoring the Database Server	33-6
What Is the Message Log?	33-6
What Is the Console?	33-12
Monitoring with ON-Monitor	33-12
Monitoring with SMI Tables	33-13
Monitoring with onstat and oncheck Utilities	33-13
Monitoring with onperf	33-13
Monitoring with the onstat Banner Line	33-14
Monitoring Configuration Information	33-15
Monitoring Checkpoint Information	33-17
Monitoring Shared Memory	33-19
Monitoring Shared-Memory Segments	33-19
Monitoring Shared-Memory Profile	33-20
Monitoring Buffers	33-21
Monitoring Buffer-Pool Activity	33-24
Monitoring Latches	33-27
Monitoring Locks	33-29
Monitoring Active Tblspaces	33-31
Monitoring Virtual Processors	33-32
Monitoring Sessions and Threads	33-35
Monitoring Parallel Database Query	33-40
Monitoring Transactions	33-45
Monitoring Databases	33-47
Monitoring Logging Activity	33-48
Monitoring Logical-Log Files	33-48
Monitoring the Physical-Log File	33-51
Monitoring the Physical-Log and Logical-Log Buffers	33-53
Monitoring Chunk Status	33-55
Monitoring the Database Server for Disabling I/O Errors	33-58
Using the Message Log to Monitor Disabling I/O Errors	33-58
Using Event Alarms to Monitor Disabling I/O Errors	33-59

Monitoring Disk Usage	33-60
Monitoring Chunks	33-60
Monitoring Tblspaces and Extents	33-64
Monitoring Sbspaces	33-67
Monitoring Simple Large Objects in a BlobSpace	33-69
Monitoring Simple Large Objects in a Dbspace	33-74
Monitoring Data-Replication Status	33-76

Section VIII Distributed Data

Chapter 34 Multiphase Commit Protocols

Two-Phase Commit Protocol	34-4
When Is the Two-Phase Commit Protocol Used?	34-4
What Goals Does the Two-Phase Commit Protocol Achieve?	34-5
Two-Phase Commit Concepts	34-5
Phases of the Two-Phase Commit Protocol	34-6
Examples of Two-Phase Commit Transactions	34-8
How the Two-Phase Commit Protocol Handles Failures	34-10
Presumed-Abort Optimization	34-17
Independent Actions	34-18
What Initiates Independent Action?	34-19
Possible Results of Independent Action	34-19
The Heuristic Rollback Scenario	34-22
The Heuristic End-Transaction Scenario	34-26
Tracking a Global Transaction	34-29
Two-Phase Commit Protocol Errors	34-29
Two-Phase Commit and Logical-Log Records	34-30
Logical-Log Records When the Transaction Commits	34-31
Logical-Log Records Written During a Heuristic Rollback	34-33
Logical-Log Records Written After a Heuristic End Transaction	34-35
Configuration Parameters Used in Two-Phase Commits	34-37
Function of the DEADLOCK_TIMEOUT Parameter	34-37
Function of the TXTIMEOUT Parameter	34-37
Heterogeneous Commit Protocol	34-38
Which Gateways Can Participate in a Heterogeneous Commit Transaction?	34-39
Enabling and Disabling Heterogeneous Commit	34-40
How Does Heterogeneous Commit Work	34-41
Implications of a Failed Heterogeneous Commit	34-43

Chapter 35	Recovering Manually from Failed Two-Phase Commit	
	Procedure to Determine If Manual Recovery Is Required	35-3
	Determine If a Transaction Was Implemented Inconsistently	35-4
	Determine If the Distributed Database Contains Inconsistent Data	35-6
	Decide If Action Is Needed to Correct the Situation	35-10
	Example of Manual Recovery	35-11

Section IX Reference

Chapter 36	ON-Monitor	
	Using ON-Monitor	36-3
	Help and Navigation Within ON-Monitor	36-4
	Executing Shell Commands from Within ON-Monitor	36-4
	ON-Monitor Screen Options	36-4
	Setting Configuration Parameters with ON-Monitor	36-12

Chapter 37	Configuration Parameters	
	ONCONFIG Parameters	37-5
	ONCONFIG File Conventions	37-6
	AFF_NPROCS	37-7
	AFF_SPROC	37-8
	ALARMPROGRAM	37-9
	BUFFERS	37-9
	CKPTINTVL	37-11
	CLEANERS	37-12
	CONSOLE	37-12
	DATASKIP	37-13
	DBSERVERALIASES	37-14
	DBSERVERNAME	37-15
	DBSPACETEMP	37-16
	DEADLOCK_TIMEOUT	37-18
	DRAUTO	37-18
	DRINTERVAL	37-19
	DRLOSTFOUND	37-20
	DRTIMEOUT	37-20
	DS_MAX_QUERIES	37-21
	DS_MAX_SCANS	37-22

DS_TOTAL_MEMORY	37-23
DUMPCNT	37-27
DUMPCORE	37-27
DUMPDIR	37-28
DUMPGCORE	37-28
DUMPSHMEM	37-29
FILLFACTOR	37-30
HETERO_COMMIT	37-30
LBU_PRESERVE	37-31
LOCKS	37-32
LOGBUFF	37-33
LOGFILES	37-34
LOGSIZE	37-35
LOGSMAX	37-36
LRUS	37-37
LRU_MAX_DIRTY	37-37
LRU_MIN_DIRTY	37-38
LTAPEBLK	37-38
LTAPEDEV	37-39
LTAPESIZE	37-40
LTXEHWM	37-41
LTXHWM	37-42
MAX_PDQPRIORITY	37-43
MIRROR	37-44
MIRROROFFSET	37-45
MIRRORPATH	37-45
MSGPATH	37-46
MULTIPROCESSOR	37-46
NETTYPE	37-47
NOAGE	37-49
NUMAIOVPS	37-50
NUMCPUVPS	37-51
OFF_RECVRY_THREADS	37-52
ON_RECVRY_THREADS	37-52
ONDBSPACEDOWN	37-53
OPCACHEMAX	37-54
OPTCOMPIND	37-54
PHYSBUFF	37-56
PHYSDBS	37-57

PHYSFILE	37-57
RA_PAGES	37-58
RA_THRESHOLD	37-59
RESIDENT	37-60
ROOTNAME	37-60
ROOTOFFSET	37-61
ROOTPATH	37-61
ROOTSIZE	37-62
SBSPACENAME	37-62
SERVERNUM	37-63
SHMADD	37-64
SHMBASE	37-65
SHMTOTAL	37-65
SHMVIRTSIZE	37-67
SINGLE_CPU_VP	37-68
STACKSIZE	37-69
STAGEBLOB	37-70
TAPEBLK	37-70
TAPEDEV	37-71
TAPESIZE	37-73
TXTIMEOUT	37-73
USEOSTIME	37-74
VPCLASS	37-75

Chapter 38 **The sysmaster Database**

What Is the sysmaster Database?	38-3
Using the System-Monitoring Interface	38-5
What Are the SMI Tables?	38-5
Accessing SMI Tables	38-6
The System-Monitoring Interface Tables	38-8
sysadinfo.	38-9
sysaudit	38-10
syschkio	38-11
syschunks.	38-11
sysconfig	38-13
sysdatabases.	38-14
sysdbslocale	38-15
sysdbspaces	38-15
sysdri	38-17
sysextents.	38-18
sysextspaces	38-18

syslocks	38-19
syslogs	38-20
sysprofile	38-21
sysptprof	38-23
sysstesprof	38-24
syssessions	38-26
syssestws	38-28
systabnames	38-29
sysvpprof	38-30
The SMI Tables Map	38-30
Using SMI Tables to Obtain onstat Information	38-33

Chapter 39

Utilities

Using the -V Option	39-3
Multibyte Characters	39-4
oncheck: Check, Repair, or Display	39-5
ondblog: Change Logging Mode	39-25
oninit: Initialize Universal Server	39-27
onlog: Display Logical-Log Contents	39-30
onmode: Mode and Shared-Memory Changes	39-35
onparams: Modify Log-Configuration Parameters	39-50
onspaces: Manage Blobspaces, Dbspaces, Extspaces, or Sbspaces	39-55
onstat: Monitor Universal Server Operation	39-75
ontape: Logging, Archives, and Restore	39-107

Chapter 40

Message-Log Messages

How the Messages Are Ordered in This Chapter	40-3
Message Categories	40-4
Messages: A-B	40-5
Messages: C	40-7
Messages: D-E-F	40-15
Messages: G-H-I	40-19
Messages: J-K-L-M	40-21
Messages: N-O-P	40-27
Messages: Q-R-S	40-34
Messages: T-U-V	40-39
Messages: W-X-Y-Z	40-42
Messages: Symbols	40-43

Chapter 41

Interpreting Logical-Log Records

Reading Logical-Log Records	41-3
Transactions That Drop a Table or Index	41-4
Transactions That Are Rolled Back	41-4
Checkpoints with Active Transactions	41-5
Distributed Transactions.	41-5
Logical-Log Record Structure	41-6
Logical-Log Record Header	41-6
Logical-Log Record Types and Additional Columns	41-7

Chapter 42

Disk Structures and Storage

Dbspace Structure and Storage	42-4
Structure of the Root Dbspace	42-4
Structure of a Regular Dbspace	42-15
Structure of a Mirrored Chunk	42-16
Structure of the Chunk Free-List Page	42-17
Structure of the Tblspace Tblspace	42-19
Structure of the Database Tblspace	42-23
Structure of a Dbspace Bit-Map Page	42-24
Structure and Allocation of an Extent	42-27
Structure and Storage of a Dbspace Page	42-34
Structure of B-Tree Index Pages	42-48
Structure of R-Tree Index Pages	42-60
Blobspace Structure and Storage	42-62
Structure of a Blobspace	42-62
Simple-Large-Object Storage and the Simple-Large-Object Descriptor	42-64
Structure of a Dbspace Blobpage	42-65
Blobspace Page Types	42-67
Structure of a Blobspace Blobpage	42-68
Sbospace Structure	42-71
Metadata Area Structure.	42-73
Sbpage Structure	42-74
A Multiple Chunk Sbospace.	42-75
Database and Table Creation: What Happens on Disk	42-76
Creating a Database	42-76
Creating a Table.	42-77

Appendix A	Files That the Database Server Uses
Appendix B	Trapping Errors
	Index

Introduction

About This Manual	3
Organization of This Manual	3
Types of Users	4
Software Dependencies	4
Assumptions About Your Locale	5
Demonstration Database	5
Major Features	6
Documentation Conventions	7
Typographical Conventions	7
Icon Conventions	8
Comment Icons	8
Feature Icons	8
Command-Line Conventions	9
How to Read a Command-Line Diagram	11
Additional Documentation	12
On-Line Manuals	12
Printed Manuals	12
Error Message Files	13
Documentation Notes, Release Notes, Machine Notes	13
Compliance with Industry Standards	14
Informix Welcomes Your Comments	14

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

The *INFORMIX-Universal Server Administrator's Guide* is a complete guide to the features that make up INFORMIX-Universal 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 Universal Server.

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

Organization of This Manual

The *INFORMIX-Universal Server Administrator's Guide* is in two volumes and contains the following sections and chapters:

- This Introduction provides an overview of the manual and describes the documentation conventions used.
- The section “What Is INFORMIX-Universal Server?” is made up of Chapters 1 and 2.
- The section “Configuration” is made up of Chapters 3 through 6.
- The section “Modes and Initialization” is made up of Chapters 7 through 9.
- The section “Disk, Memory, and Process Management” is made up of Chapters 10 through 19.

- The section “Logging and Log Administration” is made up of Chapters 20 through 26.
- The section “Fault Tolerance” is made up of Chapters 27 through 32.
- The section “Monitoring INFORMIX-Universal Server” is made up of Chapter 33.
- The section “Distributed Data” is made up of Chapters 34 and 35.
- The section “Reference” is made up of Chapters 36 through 42.
- [Appendix A](#) describes the files that the database server uses.
- [Appendix B](#) describes a method for trapping error information.

Types of Users

This manual is written for Universal Server administrators and others who need to reference administrative tasks.

If you have limited experience with relational databases, SQL, or your operating system, refer to [Getting Started with INFORMIX-Universal Server](#) for a list of introductory texts.

Software Dependencies

This manual assumes that you are using INFORMIX-Universal Server, Version 9.1, as your database server. If you are using an optical-storage subsystem for multimedia data, you access the data with the INFORMIX-OnLine/Optical product.

In this manual, all instances of Universal Server refer to INFORMIX-Universal Server.

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 [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. Sample command files are also included.

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 Appendix A of the [Informix Guide to SQL: Reference](#).

The script that you use to install the demonstration database is called **dbaccessdemo7** and is located in the **\$INFORMIXDIR/bin** directory. For a complete explanation of how to create and populate the demonstration database on your database server, refer to the [DB-Access User Manual](#).

Major Features

This section highlights the major features of INFORMIX-Universal Server, Version 9.1:

- Support for extensions to the database server in the form of DataBlade modules and user-defined routines
- Support for the storage of smart large objects (CLOB and BLOB data types) in addition to simple large objects (TEXT and BYTE data types)
- Ability to access virtual tables (extspaces), such as a file with columnar data or a spreadsheet, from within a database
- Support for user-defined access methods.
- Ability to designate and start a virtual processor in which to run user-defined routines (VPCLASS configuration parameter)
- New configuration parameters: VPCLASS and SBSPACENAME
- Ability to add new VP classes (onmode) dynamically
- Core extensibility and scalability
- Extensions to the following utilities to support Universal Server features: **oncheck**, **oninit**, **onlog**, **onmode**, **onspaces**
- Support for the communications support module, which allows a user of a client application to authenticate its identity to the server without sending data that would enable someone to impersonate the user

The Introduction to each Version 9.1 product manual contains a list of major features for that product. The Introduction to each manual in the Version 9.1 *Informix Guide to SQL* series contains a list of new SQL features.

Major features for Version 9.1 Informix products also appear in release notes.

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.
<i>italics</i>	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.



Tip: When you are instructed to “enter” characters or to “execute” a command, immediately press RETURN after the entry. When you are instructed to “type” the text or to “press” other keys, no RETURN is required.

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

Feature Icons

Feature icons identify paragraphs that contain feature-specific information.

Icon	Description
	Identifies information that relates to the Informix Global Language Support (GLS) feature.

These icons can apply to a row in a table, one or more paragraphs, or an entire section. A ♦ symbol indicates the end of the feature-specific information.

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.

Universal Server supports a variety of command-line options. Each valid command-line option is illustrated in a diagram in [Chapter 39, “Utilities.”](#)

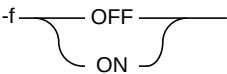
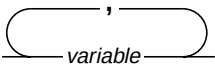
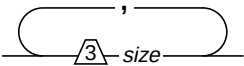
Each diagram displays the sequences of required and optional elements that are valid in a command. A diagram begins at the upper-left corner with a command. It ends at the upper-right corner with a vertical line. Between these points, you can trace any path that does not stop or back up. Each path describes 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.
<i>variable</i>	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.

(1 of 3)

Element	Description
.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 <i>Informix Migration Guide</i> . 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 <i>INFORMIX-Universal Server Archive and Backup Guide</i> . Imagine that the subdiagram is spliced into the main diagram at this point.
— ALL —	A shaded option is the default action.
→ →	Syntax within a pair of arrows indicates a subdiagram.
—	The vertical line terminates the command.

Element	Description
	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 ($\triangle 3$) 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 <i>size</i> no more than three times within this statement segment.

(3 of 3)

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

A 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 either the installation guide for your product or the installation insert that accompanies the documentation CD.

The documentation set that is provided on the CD describes Universal Server, its implementation of SQL, and its associated application-programming interfaces. For an overview of the manuals in the Universal Server documentation set, see [Getting Started with INFORMIX-Universal Server](#).

Printed Manuals

The Universal Server documentation set describes Universal Server, its implementation of SQL, and its associated application-programming interfaces. For an overview of the manuals in the Universal Server documentation set, see [Getting Started with INFORMIX-Universal Server](#).

To order printed manuals, call 1-800-331-1763 or send email to moreinfo@informix.com.

Please provide the following information:

- 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 the Informix error messages and their corrective actions. To read the error messages in the ASCII file, Informix provides scripts that let you display error messages on the screen (**finderr**) or print formatted error messages (**rofferr**). For a detailed description of these scripts, see the Introduction to the [Informix Error Messages](#) manual.

Documentation Notes, Release Notes, Machine Notes

In addition to printed documentation, the following on-line files, located in the **\$INFORMIXDIR/release/en_us/0333** directory, supplement the information in this manual.

On-Line File	Purpose
ADMINDOC_9.1	The documentation-notes file describes features that are not covered in this manual or that have been modified since publication.
SERVERS_9.1	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.
IUNIVERSAL_9.1	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.

Please examine these files because they contain vital information about application and performance issues.

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, on INFORMIX-Universal Server. In addition, many features of Universal Server 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:

415-926-6571

We appreciate your feedback.

What Is INFORMIX-Universal Server?

Section I



What Is INFORMIX-Universal Server?

What Is Universal Server?	1-3
Complex and User-Defined Data Types	1-6
Complex Data Types	1-6
User-Defined Data Types	1-6
Support for Large Objects	1-7
Smart Large Objects and Sbspaces	1-7
Support for External Spaces	1-8
Inheritance	1-8
User-Defined Routines and DataBlade Modules	1-8
User-Defined Access Methods	1-9
Universal Server Features	1-9
Client / Server Architecture	1-10
The Client / Server Connection	1-10
Scalability.	1-11
High Performance.	1-11
Raw Disk Management.	1-12
Dynamic Shared-Memory Management	1-12
Dynamic Thread Allocation	1-13
Parallelization	1-13
Fault Tolerance and High Availability	1-13
Dbospace and Logical-Log Backups of Transaction Records	1-14
Fast Recovery	1-14
Mirroring	1-15
High-Availability Data Replication.	1-15
Point-in-Time Recovery	1-15
Distributed Data Queries	1-16
Database Server Security	1-16
Who Uses Universal Server?	1-17
Database Users	1-17
Database Administrators	1-17

Universal Server Administrators	1-18
Application Developers	1-18
Features Beyond the Scope of Universal Server	1-18
No Bad-Sector Mapping	1-19
No Scanning or Compression of Simple Large Objects	1-19

T

his chapter introduces INFORMIX-Universal Server. It includes the following sections:

- What is Universal Server?
- Who uses Universal Server?
- Universal Server features
- Features beyond the scope of Universal Server

These sections provide an overview of Universal Server and direct you to more detailed information, either in this document or in other documents.

What Is Universal Server?

Universal Server is a *database server*. A database server is software that manages access to databases for client applications. It is the principal component of a database-management system.

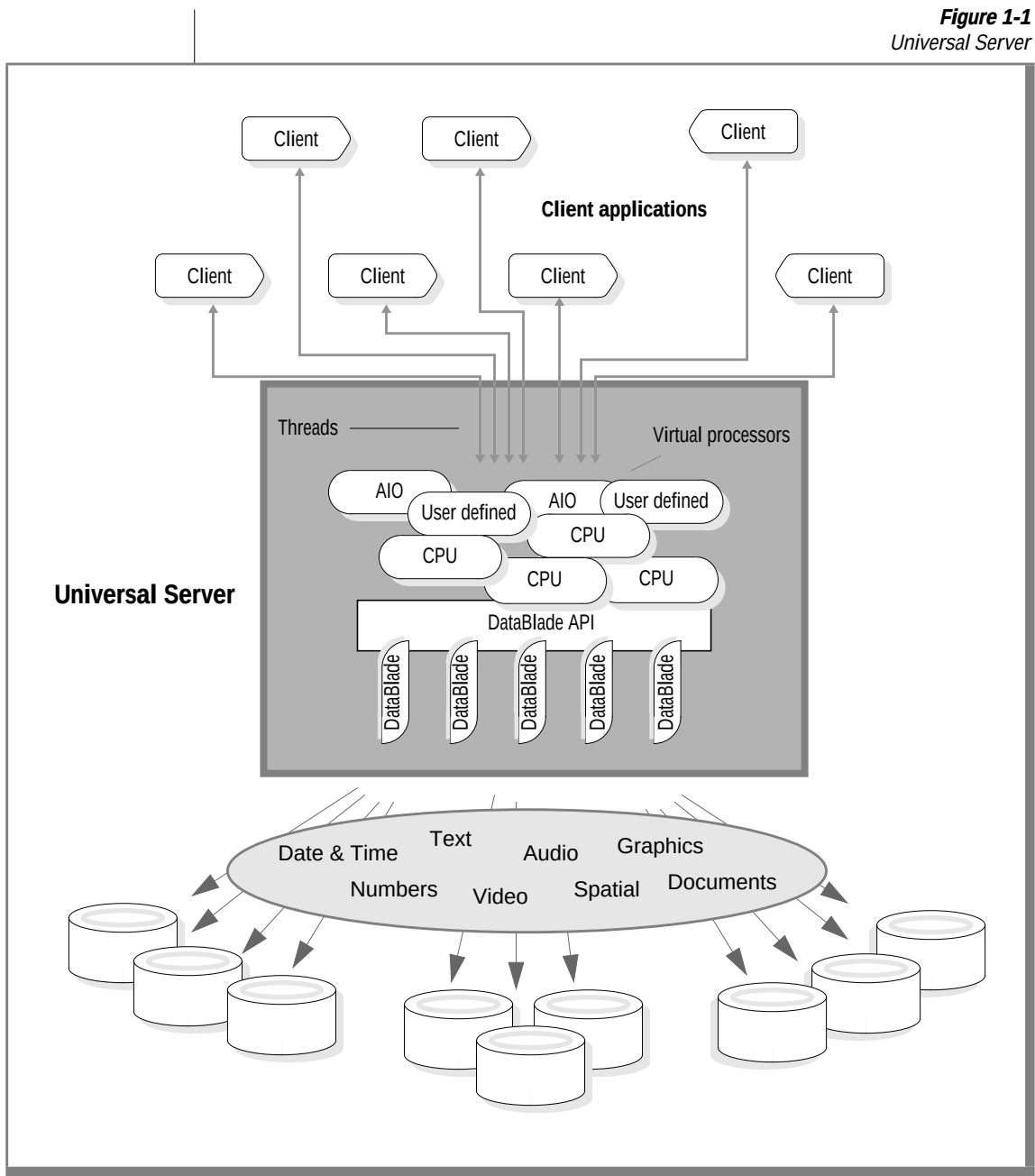
Universal Server is specifically a database server in an *object-relational database-management system* (ORDBMS). An object-relational database is organized in tables that consist of rows and columns, just as a relational database is. However, Universal Server also supports nonalphanumeric data types (objects) and enables you add new data types and supporting routines to *extend* its data management capabilities.

Universal Server supports the following types of data and extensibility:

- Built-in data types
- Complex data types
- User-defined data types
- Type and table inheritance
- User-defined casting of data types
- Storage and access of large objects
- User-defined routines and DataBlade modules (pre-packaged data types and supporting routines)
- User-defined access methods

[Figure 1-1 on page 1-5](#) illustrates the major components and the extensibility of Universal Server.

Figure 1-1
Universal Server



The following sections describe in more detail the support that Universal Server provides for new data types.

Complex and User-Defined Data Types

Relational database management systems traditionally have supported alphanumeric data such as fixed-length or variable-length character strings, integers, decimal and floating-point numbers, date, time, and time-interval structures. Universal Server provides *built-in* support for these data types.

Universal Server also inherently supports complex data types, however, and allows you to define and implement new data types. You can use all Universal Server data types, including user-defined data types, equally. You can use all data types in SQL statements, to define columns in tables, and as function arguments. For information about the Universal Server built-in data types, refer to the [Informix Guide to SQL: Reference](#).

Complex Data Types

Universal Server enables you to build complex data types from a combination of built-in, user-defined, or other complex data types. In contrast to user-defined *opaque types*, whose characteristics Universal Server knows nothing about, you can access the components of a complex data type using SQL.

For more information on complex data types, refer to Chapter 3 of the [Informix Guide to SQL: Reference](#), Chapter 10 of the [Informix Guide to SQL: Tutorial](#), and the Data Types segment in the [Informix Guide to SQL: Syntax](#).

User-Defined Data Types

Universal Server provides the ability to support additional data types such as video, audio, photographs, documents, two-dimensional spatial objects, and so on. You can purchase DataBlade modules, from Informix or third-party vendors, to support many such data types. Universal Server also enables you to develop your own user-defined routines and access methods to support new data types.

For information on implementing new data types, refer to [Extending INFORMIX-Universal Server: Data Types](#). For an introduction and references to more information on user-defined routines, see [“User-Defined Routines and DataBlade Modules” on page 1-8](#).

Support for Large Objects

Relational systems have provided the ability to store large chunks of non-alphanumeric data as binary or text data types. Client applications, however, were left with the problem of accessing and manipulating the content of these objects after the database server retrieved them.

Universal Server supports *simple large objects* and *smart large objects* for storing large data objects in a database. Simple large objects are the TEXT and BYTE data types that INFORMIX-OnLine Dynamic Server also supports. Smart large objects are the Character Large Object (CLOB) and Binary Large Object (BLOB) data types that only Universal Server supports. Smart large objects provide the ability to seek, read from and write to segments of the object. Smart large objects are also recoverable and obey transaction isolation modes.

Smart Large Objects and Sbspaces

You can use smart large objects to store user-defined data types such as video and audio clips, pictures, large text documents, and spatial objects such as drawings and maps. Universal Server stores large objects in *sbspaces*. Sbspaces are collections of Universal Server storage spaces that exclusively store smart large objects. For information on sbspaces, see [“What Is an Sbspace?” on page 14-24](#). For information on how to create an sbspace, see [“onspaces: Manage Blobspaces, Dbbspaces, Extbspaces, or Sbspaces” on page 39-55](#).

INFORMIX-ESQL/C provides client applications with the Smart Large Object application-programmer’s interface (API), which enables an application to access a smart large object much like an operating-system file. The API enables you to access any portion of a smart large object. For information on how to access either a simple large object or a smart large object from a client application, refer to the [INFORMIX-ESQL/C Programmer’s Manual](#).

For information on how to assign a CLOB or BLOB column to an sbspace, refer to the CREATE TABLE statement in the [Informix Guide to SQL: Syntax](#).

Support for External Spaces

Universal Server supports *ext spaces* (*external spaces*), which are storage spaces that Universal Server does not manage directly. You can specify an external space as the storage space for a table or index for which you create an access method. An extspace is a description of an external location. It provides a way to associate a logical name with a location string such as a pathname or network gateway, for example.

For more information on extspaces, refer to [“What Is a Extspace?” on page 14-25](#). For information on creating and dropping extspaces, refer to [“onspaces: Manage Blobspaces, Dbspaces, Extspaces, or Sbspaces” on page 39-55](#).

For information on specifying an external space as the storage location for a table or index, see the CREATE TABLE and CREATE INDEX statements in the [Informix Guide to SQL: Syntax](#). For more information on user-defined access methods, refer to [“User-Defined Access Methods” on page 1-9](#).

Inheritance

Universal Server supports *inheritance*, which allows named row types and typed tables to inherit representation and behavior from parent types and tables.

For more information on inheritance, see Chapter 10 of the [Informix Guide to SQL: Tutorial](#) and the CREATE TABLE statement in the [Informix Guide to SQL: Syntax](#).

User-Defined Routines and DataBlade Modules

You can extend Universal Server data management capabilities by either using the features that enable you to add data types, supporting routines, and access methods or by adding a DataBlade module. A *DataBlade module* is a pre-packaged custom data type and all the required supporting routines. You can obtain DataBlades from Informix and from third-party vendors. You can also create your own DataBlade module using the DataBlade Developers Kit.

For information on implementing user-defined routines, see [Extending INFORMIX-Universal Server: User-Defined Routines](#). For information on creating a DataBlade module, refer to [DataBlade API Programmer's Manual](#) and [Datablade Developer Kit User's Guide](#). The DataBlade Developers Kit is a separate Informix product. It is not included with Universal Server.

User-Defined Access Methods

In addition to adding user-defined data types and user-defined routines, you can also create new access methods, if necessary, to support new data types or to include in a database information that is stored external to Universal Server. For information on creating a new access method, see the *Virtual-Table Interface Programmer's Manual*.

Universal Server Features

In addition to its ORDBMS characteristics, Universal Server incorporates the following features:

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

Each of these features is explained in the following sections.

Client/Server Architecture

Universal Server is a *server* for client applications. Specifically, Universal Server is a *database server* that processes requests for data from client applications. It accesses the requested information from its databases, if possible, and sends back the results. 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.

The *client* is an application program that a user runs to request information from a database. Client applications use Structured Query Language (SQL) to send requests for data to Universal 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 processes are independent of Universal Server processes. Database users run client applications as the need to access information arises. The database administrator starts the Universal Server processes by executing the **oninit** utility. Universal Server processes are presumed to execute continuously during the period that users access the databases. For a description of the processes and the methods by which they serve client applications, refer to [Chapter 10, “What Is the Universal Server Architecture?”](#)

The Client/Server Connection

A client application communicates with Universal Server through the connection facilities that the database server provides. These facilities are fully described in [Chapter 4, “Client / Server Communications.”](#)

At the source-code level, a client connects to Universal Server through an SQL statement. Beyond that, the client’s use of connection facilities is transparent to the application. Library functions that are automatically included when a client program is compiled enable the client to connect to Universal Server.

As the database administrator, you specify the types of connections that Universal Server supports in a connectivity-information file called **sqlhosts**. The **sqlhosts** file contains the names of each of the database servers (called *dbservernames*), and any aliases, to which the clients on a host computer can connect. For each *dbservername* and each alias, you specify the protocol that a client must use to connect to that database server. When the client connects to Universal Server through an SQL statement, the client transparently accesses this information and makes the connection using the specified protocol.

Scalability

The Universal Server *dynamic scalable architecture* (DSA) enables you to add both processes and shared memory while Universal Server is in on-line mode. The Universal Server dynamic-scalable architecture is described in [Chapter 10, “What Is the Universal Server Architecture?”](#) [Chapter 12, “Shared Memory,”](#) describes how Universal Server uses shared memory. For tuning and performance information, refer to the [INFORMIX-Universal Server Performance Guide](#).

High Performance

Universal 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.

Raw Disk Management

Universal Server can use both UNIX file-system disk space and raw disk space. When Universal Server uses raw disk space, however, it performs its own disk management using raw devices. By storing tables on one or more raw devices instead of in a standard operating-system file system, Universal Server can manage the physical organization of data and minimize disk I/O. Doing so results in three performance advantages:

- Universal Server is not restricted by operating-system limits on the number of tables that can be accessed concurrently.
- Universal Server optimizes table access by guaranteeing that rows are stored contiguously.
- Universal Server eliminates operating-system I/O overhead by performing direct data transfer between disk and shared memory.

If these issues are not a primary concern, you can also configure Universal Server to use regular operating-system files to store data. In this case, Universal Server manages the file contents, but the operating system manages the I/O. For more information about how Universal Server uses disk space, refer to [Chapter 14, “Where Is Data Stored?”](#)

Dynamic Shared-Memory Management

All applications that use a single instance of a Universal Server 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. Disk access, and the corresponding degradation in performance, might not occur. Through shared-memory management, Universal Server minimizes disk access and the associated impact on performance.

Universal Server 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. Universal 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 13, “Managing Shared Memory.”](#)

Dynamic Thread Allocation

Universal 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, Universal Server provides a flexible architecture that is well suited for both on-line transaction processing (OLTP) and for decision-support applications. For a description of Universal Server dynamic scalable architecture, refer to [Chapter 10, “What Is the Universal Server Architecture?”](#)

Parallelization

Universal Server can allocate multiple threads to work in parallel on a single query. This feature is known as the parallel database query (PDQ) feature. Universal Server allows a table to be fragmented over multiple disks according to a distribution scheme. This feature is known as fragmentation. The PDQ feature is most effective when you use it with the fragmentation feature. For a description of the PDQ feature, refer to [Chapter 18, “What Is PDQ?”](#) For a description of fragmentation, refer to [Chapter 16, “What Is Fragmentation?”](#)

Fault Tolerance and High Availability

Universal 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

Dbospace and Logical-Log Backups of Transaction Records

Universal 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 [INFORMIX-Universal Server Archive and Backup Guide](#), Universal Server allows you to create backup tapes and logical-log backup tapes while users are accessing Universal Server. 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 Universal 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 Universal Server starts up, it checks if the physical log is empty because that implies that Universal Server shut down in a controlled fashion. If the physical log is *not* empty, Universal Server automatically performs an operation called *fast recovery*. Fast recovery automatically restores Universal Server databases to a state of physical and logical consistency after a system failure that might leave one or more transactions uncommitted. During fast recovery, Universal 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

Universal Server spawns multiple threads to work in parallel during fast recovery. Fast recovery is explained in detail in [Chapter 26, “What Is Fast Recovery?”](#)

Mirroring

When you use disk mirroring, Universal 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 27, “What Is Mirroring?”](#)

High-Availability Data Replication

If your organization requires a high degree of availability, you can replicate Universal Server and its databases, running simultaneously on a second computer. Replicating Universal 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 29, “What Is 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

Universal 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 the [INFORMIX-Universal Server Backup and Restore Guide](#) for more information about this feature. If you use ON-Archive, refer to the [INFORMIX-Universal Server Archive and Backup Guide](#).

Distributed Data Queries

Universal 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 34, “Multiphase Commit Protocols.”](#)

You can also use Universal Server in a heterogeneous environment that conforms to X / Open. For more information about using Universal Server within an X / Open environment, refer to the [Informix Embedded SQL TP/XA Programmer's Manual](#).

Database Server Security

Universal 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, Universal Server offers the ability to audit database events on a database-server-wide basis. Auditing, described in the [INFORMIX-Universal Server 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.

Who Uses Universal Server?

Universal Server administrators understand that Universal Server combines fault-tolerant, OLTP performance with multimedia capabilities and a dynamic architecture to take advantage of available hardware resources, but not all users of Universal Server understand it in that way. The question “What is Universal Server?” means different things to different users. The following types of individuals who interact with Universal Server all have a different perspective:

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

Database Users

Database users access, insert, update, and manage information in databases using *Structured Query Language* (SQL), often embedded in a client application. These users might be completely unaware that they are using Universal Server. To them, Universal Server is a nameless aspect of the system that they are using.

Database Administrators

The *database administrator* (DBA) is primarily responsible for creating and managing access control for a database, as described in [“Database Server Security” on page 1-16](#). The DBA uses SQL statements to grant and revoke privileges to ensure that the correct individuals are able to perform the actions they need to and that untrained or unscrupulous users are kept from performing potentially damaging or inappropriate activities. The [Informix Guide to SQL: Tutorial](#) and the [Informix Guide to SQL: Syntax](#) are primary sources of information for the DBA.

Universal Server Administrators

In contrast to the DBA, a Universal Server administrator administers, maintains, and operates Universal Server. The tasks involved in Universal Server administration are described in [Chapter 2, “Overview of Administration.”](#)

Application Developers

For the developers of client applications, Detachable modules, and user-defined routines, Universal Server is a database server that offers a number of possibilities for data management, multimedia, isolation levels, and so on. Universal Server can integrate information objects such as scanned and digitized images, voice, graphs, facsimiles, and word-processing documents into an SQL-based object-relational database.

The concepts of object-relational databases managed by Informix database servers are explained in the [Informix Guide to SQL: Tutorial](#). The [Informix Guide to SQL: Reference](#) and the [Informix Guide to SQL: Syntax](#), provide information on using Structured Query Language (SQL). For information on developing C-language database applications, refer to the [INFORMIX-ESQL/C Programmer's Manual](#). For additional references to Universal Server documentation, refer to [Getting Started with INFORMIX-Universal Server](#).

Features Beyond the Scope of Universal Server

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

No Bad-Sector Mapping

Universal Server relies on the operating system of your host computer for bad-sector mapping. Universal 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, Universal Server retries the access several times to ensure that the condition is not spurious. If the condition is confirmed, Universal Server marks as *down* the chunk where the read or write was attempted.

Universal 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 Universal Server detects an I/O error on a chunk that is *not* mirrored, Universal Server marks the chunk as down. If the down chunk contains logical-log files, the physical log, or the root dbspace, Universal Server immediately initiates an abort. Otherwise, Universal Server can continue to operate, but applications cannot access the down chunk until its dbspace is restored.

No Scanning or Compression of Simple Large Objects

Universal Server receives large-object 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 large objects to a database from an application program, refer to the [*INFORMIX-ESQL/C Programmer's Manual*](#).

Universal Server does not contain any mechanisms for scanning simple large objects and inserting the data into a file, or for compression, after the large object has been scanned.

Overview of Administration

Initial Tasks	2-3
Routine Tasks.	2-4
Changing Modes	2-4
Backing Up Data and Backing Up Logical-Log Files	2-4
Monitoring Activity	2-5
Checking for Consistency	2-5
Configuration Tasks	2-5
Managing Disk Space	2-5
Managing Database-Logging Status.	2-6
Managing Extensions.	2-6
Logical-Log Administration	2-6
Physical-Log Administration	2-7
Using Auditing.	2-7
Using Mirroring	2-7
Using Data Replication	2-7
Managing Shared Memory.	2-8
Managing Virtual Processors	2-8
Managing Parallel Database Query	2-8

As an INFORMIX-Universal Server database administrator, you need to be aware of the tasks and responsibilities that fall into your domain. At first glance, the tasks might appear overwhelming. As you become familiar with your database server, the areas will not seem as daunting.

This chapter describes the three types of tasks that the administration of Universal Server entails:

- Initial installation and configuration
- Routine tasks that are performed on a regular basis
- Configuration tasks that are performed less frequently

Initial Tasks

When you first acquire Universal Server, you need to perform some initial installation and configuration tasks. These tasks are described in [Chapter 3, “Installing and Configuring.”](#)

If you are moving to Universal Server from INFORMIX-OnLine Dynamic Server, refer to the [Informix Migration Guide](#).

If you are installing a DataBlade module, refer to the accompanying documentation for installation instructions. If you are moving to Universal Server from the Illustra database server, or moving a DataBlade module from the Illustra server to Universal Server, refer to the *Illustra to INFORMIX-Universal Server Migration Guide*.

You must also configure connectivity for Universal Server and your client applications, as explained in [Chapter 4, “Client/Server Communications.”](#)

These tasks can seem complicated and time consuming. Fortunately, they are not common tasks and are not representative of most of the administrative work that Universal Server needs.

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 Universal Server database server is available 24 hours a day, 7 days a week, you might not bring Universal Server to off-line mode, so mode changes would not be a routine task.

Changing Modes

The Universal Server administrator is responsible for starting up and shutting down Universal Server, as well as for placing it in administration mode when necessary.

[Chapter 7, “What Are the Operating Modes?”](#) describes each Universal Server mode, and [Chapter 8, “Managing Modes,”](#) explains how to move Universal Server from one mode to another.

Backing Up Data and Backing Up Logical-Log Files

Frequent backing up of data and logical-log files ensures that Universal 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. If you use ON-Bar for your backup and restore system, refer to [INFORMIX-Universal Server Backup and Restore Guide](#) for more information. For more information about ON-Archive or the **ontape** utility, refer to the [INFORMIX-Universal Server Archive and Backup Guide](#).

Monitoring Activity

Universal Server design enables you to monitor every aspect of operation. [Chapter 33, “Monitoring the Database Server,”](#) provides you with descriptions of the available information, instructions for how to obtain it, and suggestions for its use. As a result of monitoring, you might need to change your configuration in one of the ways described in [“Configuration Tasks.”](#)

Checking for Consistency

Informix recommends that you perform occasional checks for data consistency. [Chapter 31, “What Is Consistency Checking?”](#) describes these checks.

Configuration Tasks

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

Managing Disk Space

If you plan to use more than one Universal Server instance on the same computer, be aware of the issues explained in [Chapter 15, “Managing Disk Space.”](#)

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

[Chapter 14, “Where Is Data Stored?”](#) explains the advantages and drawbacks of different disk configurations. [Chapter 15, “Managing Disk Space,”](#) describes the actual disk-management tasks.

Finally, [Chapter 16, “What Is Fragmentation?”](#) describes how to fragment tables and indexes over multiple disks to improve performance, concurrency, data availability, and backups.

Managing Database-Logging Status

As a Universal Server administrator, you can control whether a database managed by your Universal Server database server 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 20, “What Is Logging?”](#) Information on how to change logging options is in [Chapter 21, “Managing Database-Logging Status.”](#)

Managing Extensions

Because you can extend the data management capabilities of Universal Server with new data types and user-defined routines and access methods, you might be required to add user-defined routines or DataBlade modules to the database server. You might also be required to add virtual processors to run new user-defined routines. For more information about creating new data types, refer to [Extending INFORMIX-Universal Server: Data Types](#). For information about installing a DataBlade module, refer to the accompanying documentation. For information about adding user-defined routines and virtual processors, refer to [Chapter 10, “What Is the Universal Server Architecture?”](#)

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 explained in [Chapter 22, “What Is the Logical Log?”](#)

Instructions for creating and modifying the logical-log configuration are provided in [Chapter 23, “Managing Logical-Log Files.”](#)

Information on backing up the logical log is included in the manuals that document the backup and restore systems. If you use ON-Bar for your backup and restore tool, refer to the [INFORMIX-Universal Server Backup and Restore Guide](#). For more information about ON-Archive or **ontape**, refer to the [INFORMIX-Universal Server Archive and Backup Guide](#).

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 25, “Managing the Physical Log.”](#)

Using Auditing

If you use Universal 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 the [INFORMIX-Universal Server Trusted Facility Manual](#).

Using Mirroring

Mirroring is described in [Chapter 27, “What Is Mirroring?”](#) Informix recommends that you mirror at least your root dbspace. Instructions for mirroring are provided in [Chapter 28, “Using Mirroring.”](#)

Using Data Replication

Data replication uses additional hardware to provide a very high degree of availability. Data replication is described in [Chapter 29, “What Is Data Replication?”](#) If you plan to use data replication, also see [Chapter 30, “Using Data Replication.”](#)

Managing Shared Memory

Managing the use of shared memory is a broad task that falls under the responsibility of the Universal Server administrator. Use of shared memory is described in [Chapter 12, “Shared Memory.”](#)

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

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

- Add segments to virtual shared memory

[Chapter 13, “Managing Shared Memory,”](#) describes the procedures to manage shared memory.

Managing Virtual Processors

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

[Chapter 10, “What Is the Universal Server Architecture?”](#) explains what virtual processors are, and [Chapter 11, “Managing Virtual Processors,”](#) explains how to change the virtual-processor configuration.

Managing Parallel Database Query

You can control the resources that Universal Server 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. [Chapter 18, “What Is PDQ?”](#) describes the parallel database query feature. [Chapter 19, “Managing PDQ and Decision Support,”](#) describes how to manage Universal Server resources for PDQ.

Configuration



Installing and Configuring

Planning	3-4
Consider Your Priorities.	3-4
Consider Your Resources	3-5
Administering	3-6
Installing	3-7
Installing When No Other Informix Products Are Present	3-7
Installing When Other Informix Products Are Present	3-8
Upgrading a Version of INFORMIX-OnLine.	3-8
Creating a Private Installation.	3-9
Running Multiple Universal Server Database Servers	3-10
Configuration Overview	3-10
Configuration Files	3-11
The onconfig.std File	3-11
The sqlhosts File	3-11
Environment Variables That Universal Server Uses	3-11
Configuring Universal Server	3-13
Set Environment Variables	3-13
Prepare the ONCONFIG Configuration File	3-14
Overview of Configuration Parameters	3-15
Root Dbspace	3-15
Other Space Management Parameters	3-16
Identification Parameters	3-17
Mirroring	3-17
Logical Logging	3-18
Physical Logging	3-19
Archiving and Logical-Log Backups	3-19
Message Logs	3-20
Shared-Memory Parameters	3-21
Shared-Memory Buffer Control	3-22

Shared-Memory Size Allocation	3-22
Fragmentation and Parallel Data Query Parameters	3-23
Virtual Processor Parameters	3-24
Time Intervals	3-25
Restores	3-25
Data Replication	3-25
Error Dumps	3-26
Event Alarms	3-26
OnLine / Optical	3-26
Allocate Disk Space	3-26
Prepare the Connectivity File	3-27
Prepare the ON-Archive Configuration File	3-27
Prepare for Global Language Support	3-27
Evaluate UNIX Kernel Parameters	3-27
Start the Database Server and Initialize Disk Space	3-28
Create Dbspaces, Sbspaces, and Blobspaces	3-28
Configure Extspaces	3-29
Perform Administrative Tasks.	3-29
Prepare UNIX Startup and Shutdown Scripts	3-29
Warn UNIX System Administrator About cron Jobs	3-31
Make Arrangements for Tape Management.	3-31
Make Sure Users Have the Correct Environment Variables	3-31

Implementing an INFORMIX-Universal Server 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 Universal Server can greatly affect the performance of database operations.

You can customize Universal Server so that it functions optimally in your particular data-processing environment. For example, a Universal Server instance that serves 1,000 users who execute frequent, short transactions is quite different from the instance where a few users make long and complicated searches.

The purpose of this chapter is to provide a point of orientation for the process of installing and configuring Universal Server. It discusses some of the issues you must consider before you install the database server, and it introduces terminology. For the specific steps to install Universal Server from the distribution media, refer to the *[INFORMIX-Universal Server Installation Guide](#)*. The following topics appear in this chapter:

- Planning
- Installing
- Configuration overview
- Configuring

The sections on installation include special instructions for installing Universal Server when other Informix products are already installed on your computer.

The sections on the configuration files help you decide which topics are most crucial for your particular environment and which topics can be deferred until you are tuning the performance of Universal Server. This chapter refers you to more detailed discussions in the rest of the book.

Planning

When you are planning for Universal Server, you need to consider both your priorities and your resources.

Consider Your Priorities

As you prepare the initial configuration and plan your backup and archiving strategies, consider your database server requirements, such as:

- 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 this Universal Server instance be used by applications on other computers?
- 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 disk space, or CPUs or availability of operators?
- How much does the database server need to do without supervision?

Consider Your Resources

Before you start the initial 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? What is the disk-controller configuration?
During the initialization of Universal Server, everything—tables, log files, indexes, data—goes into 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. The management of disk space is discussed in [Chapter 15, “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. Universal Server keeps statistics that help you adjust these parameters after your Universal Server database server 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 [INFORMIX-Universal Server Backup and Restore Guide](#) for more information. For information on creating an archive, refer to the [INFORMIX-Universal Server Archive and Backup Guide](#).
- What are the UNIX kernel parameters? How much shared memory is available? How much of it can you use for Universal Server?
You might need the assistance of the UNIX system administrator to evaluate and modify the UNIX kernel parameters.
- Are you installing any DataBlade modules? How much shared memory will each of them require?
You install DataBlade modules after you have installed Universal Server. You might want to consider the shared-memory requirements of your DataBlade modules, however, in determining the shared-memory requirements of the database server. For information about the impact of DataBlade modules on shared-memory requirements, refer to the machine-notes file. For information on the name and location of the machine-notes file, refer to [“Documentation Notes, Release Notes, Machine Notes” on page 13.](#)

- What are the network names and addresses of the other computers on your network? Does your system run Network Information Service (NIS)?

You might need the assistance of the network administrator to update the operating-system network files.

Administering

Universal Server provides the following administrative tools to create configuration files, change modes, change various aspects of your configuration, and monitor statistics about the database server:

- A text editor
- ON-Monitor
- ON-Bar or ON-Archive or the **ontape** utility
- ON-Audit
- Universal Server utilities

ON-Monitor and the Universal Server utilities are discussed in [Chapter 36, “ON-Monitor,”](#) and [Chapter 39, “Utilities.”](#) The ON-Bar backup and restore system is described in [INFORMIX-Universal Server Backup and Restore Guide](#). ON-Archive and **ontape** are discussed in the [INFORMIX-Universal Server Archive and Backup Guide](#), and ON-Audit is discussed in the [INFORMIX-Universal Server Trusted Facility Manual](#).

You use the following tools to monitor the behavior of the database server:

- The message log
- The **sysmaster** database
- Console

The message log and the **sysmaster** database are discussed in [Chapter 40, “Message-Log Messages,”](#) and [Chapter 38, “The sysmaster Database.”](#)

Installing

Installation refers to the process of loading the product files onto your UNIX system and running the installation script to set up the product files correctly. Some of the specific steps that you should follow depend on your environment. The next several sections cover installation of Universal Server for the following environments:

- Installing Universal Server when no other Informix products are present
- Installing when other Informix products are present
- Upgrading a version of INFORMIX-OnLine Dynamic Server
- Creating a Private Installation
- Running multiple Universal Server database servers on one host computer

Installing When No Other Informix Products Are Present

The [INFORMIX-Universal Server Installation Guide](#) gives complete instructions for installing Universal Server when no other Informix products are already installed on your computer. For instructions, refer to the [INFORMIX-Universal Server Installation Guide](#).

Installing When Other Informix Products Are Present

If you are installing several Informix products, you must install them in this order:

1. SQL application-programming interfaces (APIs), such as INFORMIX-ESQL / C
2. Universal Server
3. DataBlade modules for Universal Server

The [INFORMIX-Universal Server Installation Guide](#) gives additional instructions for some of these products. The individual product guides provide any additional information that might be required. Please refer to those guides.

Upgrading a Version of INFORMIX-OnLine

If you are upgrading to INFORMIX-Universal Server from a version of INFORMIX-OnLine, refer to the [Informix Migration Guide](#) for specific instructions.

Important: *You can migrate directly to Universal Server only from INFORMIX-OnLine Dynamic Server, Version 7.2 or later.*

In general, when you migrate to Universal Server, you must perform the following tasks before installation:

1. Decide where to store the Universal Server files.

When you install Universal Server, it overwrites any files associated with OnLine that might exist in the **\$INFORMIXDIR** directory. If you want to preserve your OnLine files, you should create a new directory for the Universal Server product.

2. Make sure you have enough space for the Universal Server.

For information about space requirements, refer to the [Informix Migration Guide](#).



3. Protect your OnLine data.
To protect your OnLine data, run consistency checks, and create a level-0 backup. If you use ON-Bar for your backup and restore tool, refer to the *INFORMIX-OnLine Dynamic Server Backup and Restore Guide* for information on how to perform a backup. If you use ON-Archive or **ontape**, refer to the archiving instructions in the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*.
4. Save a copy of the OnLine configuration file. Print it out to use as you make your new configuration file.
5. Take OnLine off-line.
6. Verify that you are logged in as user **root**. Make sure that your **INFORMIXDIR** environment variable is set to the directory where Universal Server will be installed. Environment variables are discussed in Chapter 4 of the *Informix Guide to SQL: Reference*.
7. Follow the instructions in the *INFORMIX-Universal Server Installation Guide*, but do not re-create the **informix** user and directory. The script installs Universal Server in the **\$INFORMIXDIR** directory specified for user **root**. The installation script does *not* bring Universal Server on-line.
8. If you are installing DataBlade modules for Universal Server, refer to the documentation for the each DataBlade module for instructions on installing and configuring it.



Warning: Do not initialize disk space when you upgrade from *INFORMIX-OnLine Dynamic Server* to *Universal Server*. If you initialize disk space, you destroy all your existing data.

Creating a Private Installation

If you are developing DataBlade modules or user-defined routines, you can create a private installation of Universal Server to facilitate debugging your code. A private installation enables you to test code extensions to Universal Server without affecting the performance of the database server or the work of other users.

Normally, Universal Server runs with the privileges of the user **informix**. Therefore, to permit a nonprivileged user to create new functions, dynamically link them with the server, and execute them, would violate security. A private installation enables you to run Universal Server using your own user ID.

You create a private installation of Universal Server using the **installonline** script. For more information on using **installonline** to create a private installation, and on the requirements and restrictions that apply to a private installation, refer to the [INFORMIX-Universal Server Installation Guide](#).

Running Multiple Universal Server Database Servers

When more than one instance of the database server runs on the same host computer, it is called *multiple residency*. To prepare to use multiple instances of Universal Server, first install and configure *one* Universal Server following the instructions in this chapter. 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 Universal Server more than once from the install media.*

Configuration Overview

After Universal Server is installed, it must be configured before it can be brought on-line. *Configuration* refers to setting specific parameters that customize the database server for your data-processing environment: quantity of data, number of tables, types of data, hardware, number of users, and security needs. Choosing appropriate configuration parameters is one of the major topics of this book.

This section introduces the files, environment variables, and utilities that the database server uses. It provides a foundation for more detailed information about configuration that is discussed throughout this book.

Configuration Files

A summary of the files that Universal Server uses appears in Appendix A. The two files that are used in configuring all Universal Server database servers are **onconfig.std** and **sqlhosts**.

The onconfig.std File

The **\$INFORMIXDIR/etc/onconfig.std** file is the *configuration file template*. It is loaded into the **\$INFORMIXDIR/etc** directory during the installation procedure. The **onconfig.std** file contains initial settings for the configuration parameters and serves as the template for all other configuration files that you create. A sample **onconfig.std** file is shown on [page A-14](#).

The sqlhosts File

The **\$INFORMIXDIR/etc/sqlhosts** file is the *connectivity file*. It describes the connections that client applications can use to connect to any of the database servers that the file describes. Each entry specifies the database server name, the type of connection, the name of the host computer, and the service name. If you define more than one type of connection for a Universal Server instance, you must specify an alias for the database server name on each entry subsequent to the first one.

You must prepare the **sqlhosts** file even if both the client application and the database server are on the same computer. The **sqlhosts** file is covered in detail in “[The \\$INFORMIXDIR/etc/sqlhosts File](#)” on [page 4-24](#).

Environment Variables That Universal Server Uses

Environment variables are discussed in detail in Chapter 4 of the *[Informix Guide to SQL: Reference](#)*. They are also discussed in appropriate spots throughout this book.

You need to be particularly aware of the following environment variables, which must be set correctly before you can initialize Universal Server:

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

- **ONCONFIG**
- **INFORMIXSERVER**

The **INFORMIXDIR** environment variable contains the full pathname of the directory where the Informix products are installed. The **PATH** environment variable must include the directory where the Universal Server executable files are stored. You set these values during installation. The **ONCONFIG** environment variable specifies the name of the active ONCONFIG configuration file. If the **ONCONFIG** environment variable is not present, Universal Server uses configuration values from the file **\$INFORMIXDIR/etc/onconfig**. For information about the **onconfig** file, see [“onconfig” on page A-9](#).

The **INFORMIXSERVER** environment variable specifies the name of the default database server. After you prepare the ONCONFIG configuration file, set **INFORMIXSERVER** to the same value that you assigned to the DBSERVERNAME configuration parameter. Strictly speaking, this environment variable is not required for initialization. However, if **INFORMIXSERVER** is not set, Universal Server does not build the **sysmaster** tables. (For information about the **sysmaster** tables, refer to [“What Is the sysmaster Database?” on page 38-3](#).) Also, **INFORMIXSERVER** is required for the ON-Monitor and DB-Access utilities.

The following environment variables are not required for initialization, but they must be set before you can use Universal Server with an application:

- **TERM**
- **TERMCAP** or **TERMINFO** (optional)
- **INFORMIXTERM** (optional)

The **TERM**, **TERMCAP**, **TERMINFO**, and **INFORMIXTERM** environment variables specify the type of terminal interface. You might need assistance from the UNIX system administrator to set these variables because they are highly system dependent.

Configuring Universal Server

This section covers the steps that you take to prepare the Universal Server for use. It refers you to other chapters that include detailed discussions of specific topics.

The following initial configuration tasks are discussed in this chapter:

1. Set environment variables (this page)
2. Prepare the ONCONFIG configuration file ([page 3-14](#))
3. Allocate disk space ([page 3-26](#))
4. Prepare the connectivity configuration file ([page 3-27](#))
5. If you will be using ON-Archive for backups, prepare the ON-Archive configuration file ([page 3-27](#))
6. Prepare for Global Language Support ([page 3-27](#)) ♦
7. Evaluate UNIX kernel parameters ([page 3-27](#))
8. Start Universal Server and initialize disk space ([page 3-28](#))
9. Create dbspaces, sbspaces, and blobspaces, if desired ([page 3-28](#))
10. Configure extspaces ([page 3-29](#))
11. Perform administrative tasks ([page 3-29](#))

If you are installing one or more DataBlade modules, install them after you have installed Universal Server. For instructions on installing the DataBlade module, refer to the DataBlade module documentation.

Set Environment Variables

Verify that you have set the **INFORMIXDIR** and **PATH** environment variables correctly. **INFORMIXDIR** is set to the full pathname of the directory in which you installed the Universal Server product. The **PATH** environment variable should include **\$INFORMIXDIR/bin**.

After you prepare the ONCONFIG configuration file, which is discussed in the next section, set the **ONCONFIG** environment variable to the name of the configuration file. Set **INFORMIXSERVER** to the default database server name (dbservername).

Environment variables are documented in Chapter 4 of the [Informix Guide to SQL: Reference](#).

GLS

The Global Language Support (GLS) feature of Informix database servers lets you use non-English characters, monetary conventions, and collating sequences. If you are using the GLS features of the database server, you need to set GLS environment variables. GLS is discussed in the [Guide to GLS Functionality](#). ♦

Prepare the ONCONFIG Configuration File

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 36, “ON-Monitor.”](#) The descriptions of individual parameters in [Chapter 37, “Configuration Parameters,”](#) indicate whether you can use ON-Monitor to set a particular parameter and, if so, which menus lead you to the entry for it.

To prepare the ONCONFIG configuration file using a standard text editor, follow these steps:

1. Make a copy of the **\$INFORMIXDIR/etc/onconfig.std** file.
Store the new file in the **\$INFORMIXDIR/etc** directory. Do not modify **onconfig.std**. Informix suggests that you choose a filename that reflects the intended use of the configuration file (accounting, personnel, testing) and the associated server number (for example, **onconfig.acctg4**). Set your **ONCONFIG** environment variable to the name of your new file.
2. Edit your new ONCONFIG file to modify the configuration parameters that you have decided to change. For the initial configuration of the database server, you can leave most of the parameters set to their initial settings.

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

- **ROOTPATH** [page 37-61](#)
- **SERVERNUM** [page 37-63](#)
- **DBSERVERNAME** [page 37-15](#)

If you are using multiple communication protocols, you must set the following parameter:

- **DBSERVERALIASES** [page 37-14](#)

Check the following parameters if you are using the **ontape** archiving tool, or the **onunload** and **onload** utilities (If you are using ON-Archive or ON-Bar, LTAPEDEV cannot be set to **/dev/null**.):

- **TAPEDEV** [page 37-71](#)
- **LTAPEDEV** [page 37-39](#)

For information about **ontape** and ON-Archive, refer to the [INFORMIX-Universal Server Archive and Backup Guide](#). For information about ON-Bar, refer to [INFORMIX-Universal Server Backup and Restore Guide](#). For information about **onunload** and **onload**, refer to the [Informix Migration Guide](#).

Verify that the following parameters have valid pathnames:

- **MSGPATH** [page 37-46](#)
- **CONSOLE** [page 37-12](#)

After your database server is configured and running, review the configuration parameters. “[Monitoring Configuration Information](#)” on [page 33-15](#) 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 known as 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:

- **ROOTNAME** [page 37-60](#)
- **ROOTPATH** [page 37-61](#)

- **ROOTOFFSET** [page 37-61](#)
- **ROOTSIZE** [page 37-62](#)

You can choose any descriptive name for the ROOTNAME, but it is usually called **rootdbs**, which is its initial setting. The ROOTPATH is the pathname of the storage allocated to the root dbspace. Choosing and allocating the storage are discussed in [Chapter 15, “Managing Disk Space.”](#) ROOTSIZE is the amount of space allocated to the root dbspace. Choosing an appropriate size for the root dbspace is discussed in [“Calculate the Size of the Root Dbspace” on page 14-33](#). For information about when you need to set ROOTOFFSET, refer to [“Do You Need to Specify an Offset?” on page 15-6](#).

Other Space Management Parameters

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

- **DBSPACETEMP** [page 37-16](#)
- **FILLFACTOR** [page 37-30](#)
- **ONDBSPACEDOWN** [page 37-53](#)
- **SBSPACENAME** [page 37-62](#)

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 specifies how full index-pages to fill index pages when indexes are created.

ONDBSPACEDOWN defines how the database server treats a disabled dbspace that is not a critical dbspace.

SBSPACENAME specifies the name of the system default sbpace. When Universal Server writes a smart large object, it writes it to the default sbpace if no sbpace has been specified for the column.

Identification Parameters

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

- **DBSERVERNAME** [page 37-15](#)
- **DBSERVERALIASES** [page 37-14](#)
- **SERVERNUM** [page 37-63](#)

DBSERVERNAME specifies the name of the database server. The name specified in DBSERVERNAME is called the *database server name* or *dbservername*. You use the dbservername in the **\$INFORMIXDIR/etc/sqlhosts** file and with the **INFORMIXSERVER** environment variable. Client applications use the dbservername in CONNECT, DATABASE, and distributed database statements and with the **DBPATH** environment variable. DBSERVERNAME is described in “[The DBSERVERNAME Configuration Parameter](#)” on page 4-44.

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 4-45.)

The value given by SERVERNUM specifies a unique integer for each Universal Server instance. It must be unique for each database server on your local host 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

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:

- **MIRROR** [page 37-44](#)
- **MIRRORPATH** [page 37-45](#)
- **MIRROROFFSET** [page 37-45](#)

Mirroring has both positive and negative effects on performance: disk updates are slower; disk reads are faster. Mirroring is discussed in [Chapter 27, “What Is Mirroring?”](#)

Logical Logging

The logical log contains a record of changes made to a Universal Server instance. The logical-log records in the logical log are used to roll back transactions, recover from system failures, and so on. The following parameters describe logical logging:

- **LOGBUFF** [page 37-33](#)
- **LOGFILES** [page 37-34](#)
- **LOGSIZE** [page 37-35](#)
- **LOGSMAX** [page 37-36](#)
- **LTXHWM** [page 37-42](#)
- **LTXEHWM** [page 37-41](#)

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. After your database server is operative, you use the system statistics to tune the buffer size. LOGBUFF is discussed in [“Logical-Log Buffer” on page 12-26](#).

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 is the size of each logical-log file. 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 22-12](#).

The *long-transaction high-water mark* parameter, LTXHWM, 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 LTXEHW, the *long-transaction exclusive-access high-water mark*. A companion parameter, LTXEHW, is the point at which the database server takes drastic action. These parameters are described in [“Avoiding Long Transactions” on page 22-19](#).

Physical Logging

The physical log contains images of all pages (units of 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:

- **PHYSFILE** [page 37-57](#)
- **PHYSDBS** [page 37-57](#)
- **PHYSBUFF** [page 37-56](#)

PHYSFILE specifies the size of the physical log. 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. Later, you can move the physical log out of the root dbspace to improve performance. When you change PHYSFILE or PHYSDBS, you must make a level-0 dbspace backup. These parameters are discussed in [Chapter 24, “What Is Physical Logging?”](#)

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. PHYSBUFF is discussed in [“Physical-Log Buffer” on page 12-27](#).

Archiving and Logical-Log Backups

To create dbspace backups and make logical-log backups for data managed by the database server, use one of the following tools:

- **ON-Bar**
- **ON-Archive**
- **ontape**

You must choose one of these tools. ON-Bar is the most recent and most powerful backup and restore system. For more information on configuring ON-Bar, refer to the [INFORMIX-Universal Server Backup and Restore Guide](#). Refer to the [INFORMIX-Universal Server Archive and Backup Guide](#) for information on ON-Archive and the **ontape** utility.

The **ontape** utility is easier to configure but does not give as much flexibility or power as ON-Bar or ON-Archive. The archiving parameters in the ONCONFIG configuration file are used only if you decide to use the **ontape** utility (although TAPEDEV and LTAPEDEV should not be set to **/dev/null**, even if you are using ON-Bar or ON-Archive). The database server uses the following parameters for archiving and logical log backups:

- TAPEDEV [page 37-71](#)
- TAPEBLK [page 37-70](#)
- TAPESIZE [page 37-73](#)
- LTAPEDEV [page 37-39](#)
- LTAPEBLK [page 37-38](#)
- LTAPESIZE [page 37-40](#)

TAPEDEV and LTAPEDEV specify tape devices. TAPEBLK and LTAPEBLK specify the block size of the tape device. TAPESIZE and LTAPESIZE specify the maximum amount of data that should be written to each tape. These parameters are discussed in the [INFORMIX-Universal Server Backup and Restore Guide](#) and in the [INFORMIX-Universal Server Archive and Backup Guide](#).

Message Logs

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

- CONSOLE [page 37-12](#)
- MSGPATH [page 37-46](#)

CONSOLE specifies the pathname for console messages. Messages that the database server directs to the console require immediate action. The initial setting, **/dev/console**, sends messages to the system-console screen. 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. The message log is described in [“What Is the Message Log?” on page 33-6](#). The messages themselves are discussed in [Chapter 40, “Message-Log Messages.”](#)

Shared-Memory Parameters

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

- **BUFFERS** [page 37-9](#)
- **LBU_PRESERVE** [page 37-31](#)
- **LOCKS** [page 37-32](#)
- **RESIDENT** [page 37-60](#)
- **STAGEBLOB** [page 37-70](#)
- **CKPTINTVL** [page 37-11](#)

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 12-24](#).

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 22-8](#).

LOCKS specifies the maximum number of locks available to user processes during transaction processing. You can use the initial setting ([page 37-32](#)) for LOCKS.

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. Residency is discussed in [“Setting Shared-Memory Configuration Parameters” on page 13-3](#).

STACKSIZE specifies the stack size for user threads. The use of stacks is discussed in [“Stacks” on page 12-31](#).

CKPTINTVL, the checkpoint interval, is the maximum time allowed to elapse before a checkpoint. Use the initial settings for the initial configuration. Modifying CKPTINTVL to improve performance is discussed in [“Setting Configuration Parameters for the Shared-Memory Performance Options with ON-Monitor” on page 13-12](#).

Shared-Memory Buffer Control

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

- LRU [page 37-37](#)
- LRU_MAX_DIRTY [page 37-37](#)
- LRU_MIN_DIRTY [page 37-38](#)
- CLEANERS [page 37-12](#)
- RA_PAGES [page 37-58](#)
- RA_THRESHOLD [page 37-59](#)

The LRU (Least Recently Used) queues manage the shared-memory pool of pages (memory spaces) used by the database server. The LRU parameters are used for performance tuning. Use the initial settings for the initial configuration. These parameters are discussed in [“LRU Queues” on page 12-36](#). CLEANERS controls the number of threads used to flush pages to disk and return the pages to the shared-memory pool. Page cleaning is discussed in [“How the Database Server Flushes Data to Disk” on page 12-45](#).

RA_PAGES and RA_THRESHOLD control the number of disk pages that the database server reads ahead during sequential scans. These parameters are discussed in [“Configuring the Database Server to Read Ahead” on page 12-41](#).

Shared-Memory Size Allocation

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

- SHMADD [page 37-64](#)
- SHMBASE [page 37-65](#)

- [SHMTOTAL](#) [page 37-65](#)
- [SHMVIRTSIZE](#) [page 37-67](#)

SHMBASE is the shared-memory base address and is computer dependent. Do not change its value. SHMVIRTSIZE specifies the size of the first piece of memory attached by the database server. SHMADD specifies the increment of memory that is added (if possible) when the database server requests more memory. SHMTOTAL specifies the maximum amount of memory that the database server is allowed to use. These parameters are discussed in [Chapter 12, “Shared Memory.”](#)

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:

- [DATASKIP](#) [page 37-13](#)
- [DS_MAX_QUERIES](#) [page 37-21](#)
- [DS_MAX_SCANS](#) [page 37-22](#)
- [DS_TOTAL_MEMORY](#) [page 37-23](#)
- [MAX_PDQPRIORITY](#) [page 37-43](#)
- [OPTCOMPIND](#) [page 37-54](#)

Use DATASKIP to control whether the database server skips an unavailable table fragment.

DS_MAX_QUERIES, DS_MAX_SCANS, DS_TOTAL_MEMORY, and MAX_PDQPRIORITY impose limits on the amount of resources that a query can use. Strategies for improving performance using fragmentation and PDQ are discussed in the [INFORMIX-Universal Server Performance Guide](#).

OPTCOMPIND advises the optimizer on an appropriate join strategy for your applications.

Virtual Processor Parameters

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

■	<code>VPCLASS</code>	page 37-75
■	<code>AFF_NPROCS</code>	page 37-7
■	<code>AFF_SPROC</code>	page 37-8
■	<code>MULTIPROCESSOR</code>	page 37-46
■	<code>NETTYPE</code>	page 37-47
■	<code>NOAGE</code>	page 37-49
■	<code>NUMAIOVPS</code>	page 37-50
■	<code>NUMCPUVPS</code>	page 37-51
■	<code>SINGLE_CPU_VP</code>	page 37-68

`VPCLASS` allows you to specify a class of virtual processors, the number that the database server should start, the maximum number allowed, the processor affinity, and priority aging. `VPCLASS` consolidates the information provided by the `NUMCPUVPS`, `NUMAIOVPS`, `AFF_SPROC`, `AFF_NPROCS`, and `NOAGE` parameters. Informix recommends that you use the `VPCLASS` parameter instead of these other parameters.

`AFF_NPROCS` and `AFF_SPROCS` control the binding of processors on computers that support processor affinity. `MULTIPROCESSOR` specifies the appropriate type of locking. `SINGLE_CPU_VP` can specify that the database server is using only one processor and allow the database server to optimize for that situation. `NUMAIOVPS` and `NUMCPUVPS` specify the number of virtual processors for AIO and CPU class threads, respectively.

`NETTYPE` provides tuning options for each communications protocol.

`NOAGE` specifies whether priority aging should be in effect.

Time Intervals

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

- `DEADLOCK_TIMEOUT` [page 37-18](#)
- `TXTIMEOUT` [page 37-73](#)
- `USEOSTIME` [page 37-74](#)

`DEADLOCK_TIMEOUT` specifies the amount of time that the database server waits for a shared-memory resource during a distributed transaction. `TXTIMEOUT` is the amount of time a participant the database server waits to receive a *commit* instruction during a two-phase commit. These two parameters apply only to transactions that are taking place over a network. They are discussed in [“Configuration Parameters Used in Two-Phase Commits” on page 34-37](#).

`USEOSTIME` controls the granularity of the time reported by the database server.

Restores

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

- `OFF_RECVRY_THREADS` [page 37-52](#)
- `ON_RECVRY_THREADS` [page 37-52](#)

For more information, refer to [“What Threads Handle Data Replication?” on page 29-13](#).

Data Replication

The data replication parameters control the behavior of a pair of data replication servers. The following parameters refer to data replication:

- `DRAUTO` [page 37-18](#)
- `DRINTERVAL` [page 37-19](#)
- `DRLOSTFOUND` [page 37-20](#)
- `DRTIMEOUT` [page 37-20](#)

Error Dumps

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

- DUMPCNT [page 37-27](#)
- DUMPCORE [page 37-27](#)
- DUMPDIR [page 37-28](#)
- DUMPGCORE [page 37-28](#)
- DUMPSHMEM [page 37-29](#)

Event Alarms

The database server can execute a file if a noteworthy event occurs. The location of the executable file is specified by the ALARMPROGRAM parameter. See [page 37-9](#).

OnLine/Optical

The STAGEBLOB parameter is used only by INFORMIX-OnLine / Optical:

- OPCACHEMAX [page 37-54](#)
- STAGEBLOB [page 37-70](#)

For more information, refer to the [INFORMIX-OnLine/Optical User Manual](#).

Allocate Disk Space

Before you allocate the disk space, study the information about disk space in [Chapter 14, “Where Is Data Stored?”](#) For information about allocating raw disk space, refer to [“Allocating Raw Disk Space” on page 15-5](#). If you want to use cooked disk space, you can follow the instructions from [“Allocating Cooked File Space” on page 15-5](#).

Prepare the Connectivity File

The connectivity file, **\$INFORMIXDIR/etc/sqlhosts**, contains information that is required to allow an Informix client application to connect to an Informix database server, in this case Universal Server. The content of the **sqlhosts** file is covered in [“The \\$INFORMIXDIR/etc/sqlhosts File” on page 4-24](#).

You do not need to specify all possible network connections in **sqlhosts** before you initialize the database server. But to make a new connection available, you must take the database server off-line and bring it to on-line mode again.

Prepare the ON-Archive Configuration File

If you use ON-Archive to back up and restore Universal 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. The [INFORMIX-Universal Server Archive and Backup Guide](#) describes the parameters for the **config.arc** file. [Appendix A](#) describes additional ON-Archive configuration files.

GLS

Prepare for Global Language Support

The GLS feature allows you to create databases using the diacritics, collating sequence, and monetary and time conventions of the language you select. No ONCONFIG configuration parameters exist for GLS, but you must set the appropriate environment variables. GLS is discussed in detail in the [Guide to GLS Functionality](#). ♦

Evaluate UNIX Kernel Parameters

Your Universal Server product arrives with a machine-specific file that contains recommended values for UNIX kernel parameters. Compare the values in this file with your current UNIX configurations. The name of the machine-specific file is listed in [“Documentation Notes, Release Notes, Machine Notes” on page 13](#).

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 Universal Server differ significantly from your current environment, consider modifying your UNIX kernel settings. Background information that describes the role of the UNIX kernel parameters in Universal Server appears in [“UNIX Kernel Configuration Parameters” on page 13-4.](#)

Start the Database Server and Initialize Disk Space

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 Universal Server, use the command with an **-i** flag to initialize the disk space as well as to bring Universal Server into on-line mode:

```
% oninit -i
```

Warning: When you execute this command, all existing data in the database-server disk space is destroyed. Use the **-i** flag only when you are starting a new database server.

You can also initialize disk space using ON-Monitor. (For more information about ON-Monitor, refer to [Chapter 36, “ON-Monitor.”](#))

Create Dbspaces, Sbspaces, and Blobspaces

Now that Universal Server is initialized, you can create dbspaces, sbspaces, and blobspaces as desired. Dbspaces, sbspaces, and blobspaces are described in [Chapter 14, “Where Is Data Stored?”](#) The allocation and management of dbspaces, sbspaces and blobspaces are discussed in [Chapter 15, “Managing Disk Space.”](#)



Configure Extspaces

You can also configure extspaces (external spaces) that store tables or indexes that Universal Server will access through user-defined routines and access methods. For a description of extspaces and their use, refer to [Chapter 14, “Where Is Data Stored?”](#) For information about configuring and managing extspaces, refer to [Chapter 15, “Managing Disk Space.”](#)

Perform Administrative Tasks

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

- Prepare UNIX startup and shutdown scripts.
- Warn the UNIX system administrator about **cron** jobs.
- Make arrangements for tape management.
- Make sure users have the correct environment variables.

Prepare UNIX Startup and Shutdown Scripts

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

Prepare the Startup Script

To prepare the UNIX startup script, add UNIX and Universal Server utility commands to the UNIX startup script, so that the script performs the following steps:

- Set the **INFORMIXDIR** environment variable to the full pathname of the directory in which Universal Server is installed.
- Set the **PATH** environment variable to include the **\$INFORMIXDIR/bin** directory.

- Set the **ONCONFIG** environment variable to the desired configuration file.
- Set the **INFORMIXSERVER** environment variable so that the **sysmaster** database can be updated (or created, if needed).
- Execute **oninit**, which starts Universal Server and leaves it in on-line mode.

If you plan to initialize multiple versions of Universal Server (multiple residency), you must reset **ONCONFIG** and **INFORMIXSERVER** and re-execute **oninit** for each instance of Universal Server.

- If you are using ON-Archive, you might want to start **oncatlgr**. For more information about ON-Archive, refer to the [*INFORMIX-Universal Server Archive and Backup Guide*](#).

If Universal Server and versions of INFORMIX-OnLine Dynamic Server, such as Version 6.0 and Version 7.2, are installed in different directories, you must reset **INFORMIXDIR** and repeat the preceding steps for each database server that you want to install.

Prepare the Shutdown Script

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

- Set the **INFORMIXDIR** environment variable to the full pathname of the directory in which Universal Server is installed.
- Set the **PATH** environment variable to include the **\$INFORMIXDIR/bin** directory.
- Set the **ONCONFIG** environment variable to the desired configuration file.
- Execute **onmode -ky**, which initiates Immediate-Shutdown and takes Universal Server off-line.

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

If Universal Server and versions of OnLine, such as Version 6.0 and Version 7.2, are installed in different directories, you must reset **INFORMIXDIR** and repeat the preceding steps for each different version.

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

Warn UNIX System Administrator About cron Jobs

Universal Server creates the **.inf.servicename** and / or **VP.servicename.xx**C files in the **/INFORMIXTMP** directory. Some UNIX systems run **cron** jobs that routinely delete all files from the **/INFORMIXTMP** directory. For information about these files, see Appendix A.

Make Arrangements for Tape Management

When you plan your data dbspace and logical-log backup schedule, as discussed in the [INFORMIX-Universal Server 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 Chapter 4 of the [Informix Guide to SQL: Reference](#).

Each user must set the following environment variables before accessing Universal 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.

Three techniques are available for setting **INFORMIXSERVER**, **INFORMIXDIR**, **PATH**, and **ONCONFIG**:

- Ask the UNIX administrator to set these environment variables for every user during the login procedure.
- Modify the login procedures for each user so that these environment variables are set during login.
- Educate your users to set the environment variables manually every time they want to work with Universal Server.

Users might need other environment variables, such as **TERMCAP** and **LC_COLLATE**, to describe their environment fully. You can prepare an environment-configuration file, **\$INFORMIXDIR/etc/informix.rc**, which sets additional environment variables for each user. Users can override environment variables set at login time or set by **informix.rc** by using a private environment-variable file, **~/.informix**, or by setting the variable at the system prompt. The environment variable files are discussed in Chapter 4 of the *Informix Guide to SQL: Reference*.

If you use ON-Archive, users might want to set the **ARC_DEFAULT** environment variable to the filename of an alternative qualifier-defaults file. For more information about ON-Archive, refer to the *INFORMIX-Universal Server Archive and Backup Guide*.

Client/Server Communications

What Is Client/Server Architecture?	4-3
What Is a Network Protocol?	4-4
Which Network Protocols Does the Database Server Support? . . .	4-4
Which Network Protocol Should You Use?	4-5
What Is a Network Programming Interface?	4-5
Which Network Programming Interfaces Does the Database Server Support?	4-6
What Interface/Protocol Combinations Are Available on Your Platform?	4-6
What Are Communications Support Services?	4-6
What is a Communications Support Module?	4-7
How to Configure the DCE-GSS Communications Support Module	4-8
Installing and Configuring DCE Components for the DCE-GSS CSM	4-8
Describing and Specifying the CSM	4-9
What Is a Connection?	4-10
How Does a Client Application Connect to a Database Server? . . .	4-10
What Connections Does the Database Server Support?	4-10
Local Connections	4-11
Shared-Memory Connections	4-11
Stream-Pipe Connections	4-12
Advantages of Stream-Pipe Connections.	4-13
Disadvantages of Stream-Pipe Connections.	4-13
Network Connections	4-13
When Client and Server Reside On Different Computers	4-14
Local Loopback Network Connections	4-14

PC Connections	4-16
Which Protocols Does the Database Server Support for PC Communications?	4-16
Which Protocol Should You Use: TCP/IP or IPX/SPX?	4-16
Configuring the Database Server for PC Client Applications with TCP/IP	4-17
Configuring the Database Server for PC Client Applications with IPX/SPX	4-18
Supplying Configuration Information to Client-Side Administrators	4-18
Connectivity Files	4-18
Network-Configuration Files	4-19
TCP/IP Connectivity Files.	4-19
IPX/SPX Connectivity Files	4-21
Network-Security Files	4-21
The /etc/hosts.equiv and ~/.rhosts Files	4-21
The ~/.netrc File	4-22
The \$INFORMIXDIR/etc/concsm.cfg file.	4-23
The \$INFORMIXDIR/etc/sqlhosts File	4-24
The dbservername Field	4-27
The nettype Field	4-27
The hostname Field	4-29
The servicename Field	4-36
The options Field	4-38
ONCONFIG Parameters for Connectivity	4-44
The DBSERVERNAME Configuration Parameter	4-44
The DBSERVERALIASES Configuration Parameter	4-45
Environment Variables for Network Connections	4-46
Examples of Client/Server Configurations.	4-46
Using a Shared-Memory Connection	4-47
Using a Local Loopback Connection.	4-48
Using a Network Connection	4-49
Using Multiple Connection Types	4-50
Accessing Multiple Database Servers	4-52
Using the Relay Module.	4-53
A Relay Module Configuration with Three Database Servers.	4-55
Using 5.x INFORMIX-STAR or 5.x INFORMIX-NET	4-57
Using a Post-6.0 Client Application with a 5.x Database Server	4-58

T

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

- Description of connectivity files
- Description of ONCONFIG connectivity parameters
- Examples of client/server configurations

What Is Client/Server Architecture?

Informix products conform to a model of software design called *client/server*. The client/server model allows you to place an application or *client* on one computer and the database *server* on another or even the same computer. Client applications issue requests for services and data from the server. The server, in turn, provides services and data to a client based on requests from the client.

You use a *communications protocol* together with a *network programming interface* to *connect* and transfer data between the client and the server. Each of these terms is 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 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 is TLI (Transport Layer 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.

Which Network Protocols Does the Database Server Support?

Universal Server supports several network protocols, including TCP/IP and IPX/SPX. However, not all platforms support all protocols. To check which protocols are supported on your platform, follow the instructions in [“What Interface/Protocol Combinations Are Available on Your Platform?”](#) on page 4-6.

Which Network Protocol Should You Use?

When you are configuring client/server connectivity, one rule is of fundamental importance: both the client and server environments must support the same protocol if successful communication is to take place. In terms of configuring your environment for client/server communications, this means that whenever you plan to connect a client that resides on one computer to a database server on a different computer, both computers must support the protocol.

Although you can configure Universal Server to support more than one protocol (see [“Starting Multiple Listen Threads” on page 10-37](#)), consider this option only if some clients use TCP/IP and some use IPX/SPX. If all your client applications support the same protocol, say TCP/IP, configure Universal Server to support TCP/IP.

You specify which protocol Universal Server uses by setting a field in the **sqlhosts** file, as explained in [“The \\$INFORMIXDIR/etc/sqlhosts File” on page 4-24](#). If you are unsure which protocols your network or networks support, consult your network administrator.

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 servers both make use of network programming interfaces to send and receive the data according to a communications protocol. More accurately, the rules discussed in the previous section that constitute a network protocol are implemented in a network driver, which is accessed by way of a network programming interface.

As noted previously, both client and 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 server using TCP / IP with sockets on another server, or vice versa.

Which Network Programming Interfaces Does the Database Server Support?

On many platforms, Universal Server supports multiple network programming interfaces. The following section explains how to check which protocols are supported on your platform.

What Interface/Protocol Combinations Are Available on Your Platform?

To check which interface / protocol combinations Universal Server supports for your operating system, refer to [“Documentation Notes, Release Notes, Machine Notes”](#) on page 13 of the Introduction.

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 services like data compression, for example.

Authentication is the process of verifying the identity of a user or an application. The most common form of authentication, for example, is to require the user to enter a name and password to obtain access to a computer or an application. Universal Server provides a default method of authentication, which is described in the section “[Network-Security Files](#)” on page 4-21. You do not need to perform any procedures to use the default authentication policy. Universal Server uses the default authentication policy when you do not specify a communications support module.



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

What is a Communications Support Module?

Universal Server enables you to add on external communications support services in the form of a *communications support module* (CSM). Universal Server includes a CSM, called DCE-GSS, which provides access to the DCE 1.1 security service, allows a client application, on behalf of its user, to authenticate its identity to a server without sending data that would enable someone to impersonate the user. The DCE-GSS CSM does not provide other communications support services. 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 13.

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 Universal 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 file name 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.

How to Configure the DCE-GSS Communications Support Module

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

1. Install and configure the DCE components according to the third-party's documentation.
2. Describe the communications support module in the **conccsm.cfg** file. For more information on how to describe the **conccsm.cfg** file, refer to [“Describing and Specifying the CSM” on page 4-9](#).
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 4-43](#)

Installing and Configuring DCE Components for the DCE-GSS CSM

The basic steps to install and configure the DCE components are:

- Install and configure the DCE clients and servers properly.
- 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 that, for the sake of consistency, that you obtain this coordination by assigning principal names that match database user (UNIX) names and database server names. Refer to the DCE documentation for more information on assigning principal names.

- 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, refer to [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 `INFORMIXKEYTAB` 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 **`concsbm.cfg`** file. The **`concsbm.cfg`** file resides in the `$INFORMIXDIR/etc` directory by default. If you wish to store the file somewhere else, you can override the default location by setting the `INFORMIXCONCSBMCFG` environment variable to the full pathname of the new location. For information on setting the `INFORMIXCONCSBMCFG` environment variable, refer to the [Informix Guide to SQL: Reference](#).

For information on how to describe the CSM in the **`concsbm.cfg`** file, refer to “[The `\$INFORMIXDIR/etc/concsbm.cfg` file](#)” on page 4-23. For information on specifying the CSM for a connection in the **`sqlhosts`** file, refer to “[The options Field](#)” on page 4-38.



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.

How Does a Client Application Connect to a Database Server?

A client application establishes a connection with a database server by using the SQL statements `CONNECT` or `DATABASE`. 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 Connections Does the Database Server Support?

The connection between a client application and a database server is handled by functionality intrinsic to Universal Server. This functionality handles all connections between the client application and the database server. To connect to a database server (using the `CONNECT` statement, for example), the client application specifies only the name of the database server. Universal Server uses the database server name to look up information about the computer on which the database server is running and about the type of connection that should be made.

Universal Server supports the following types of connections to communicate between client applications and database servers:

- Local connections
 - Shared-memory
 - Stream-pipe
- Network connections
 - Computer-to-computer
 - Local loopback

The following sections explain how these connections work.

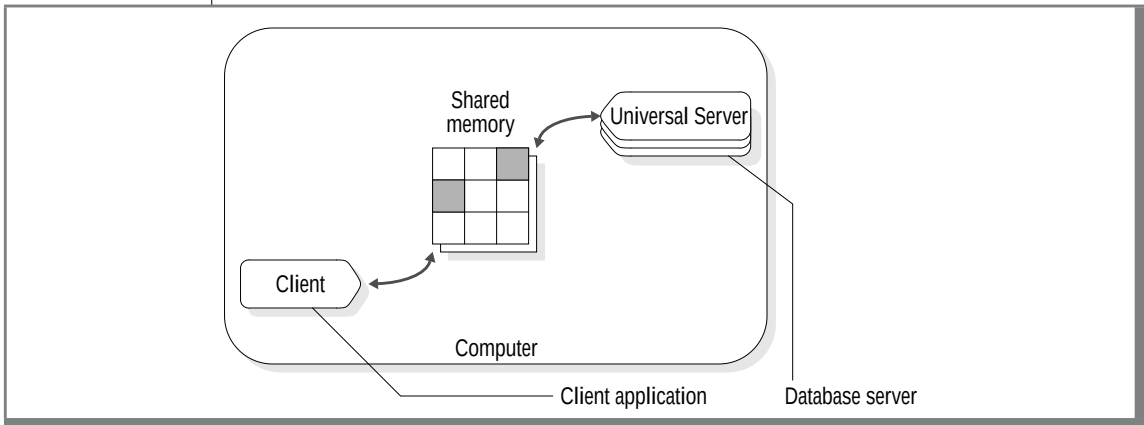
Local Connections

You can use *shared-memory* connections or *stream-pipe* connections only when your client application and the database server are on the same computer.

Shared-Memory Connections

The shared memory used for communications is not the same as the resident shared memory used by Universal Server. See [“The Communications Portion of Shared Memory” on page 12-33](#) for a discussion of shared-memory communications. [Figure 4-1 on page 4-12](#) shows 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 does explicit memory addressing or overindexes data arrays. Such errors do not affect the application if you use network communication. For more information about shared-memory communication, refer to [“Where the Client Attaches to the Communications Portion”](#) on page 12-11.

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 advantage:

- 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 might be slower than shared-memory connections on some computers.
- Stream pipes are not available on all platforms.

Network Connections

You must use a network connection when the client application resides on one computer and the database server resides on another computer. You can also use a network connection when both the client application and the database server are on the same computer. Both of these configurations are explained in the following sections.

Universal Server supports the following types of interface / protocol combinations.

Interface	Network Protocol
sockets	TCP / IP
TLI	TCP / IP
TLI	IPX / SPX

The machine-notes file that is part of your installation package tells which protocol/interface combinations are supported for your platform. (See [“Documentation Notes, Release Notes, Machine Notes”](#) on page 13 of the Introduction.)

When Client and Server Reside On Different Computers

Figure 4-2 shows a network connection where the client application resides on one computer and the database server resides on another computer.

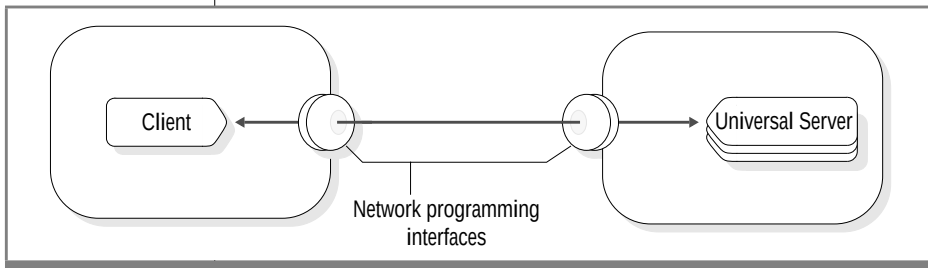


Figure 4-2
A Network Connection

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 4-3 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.

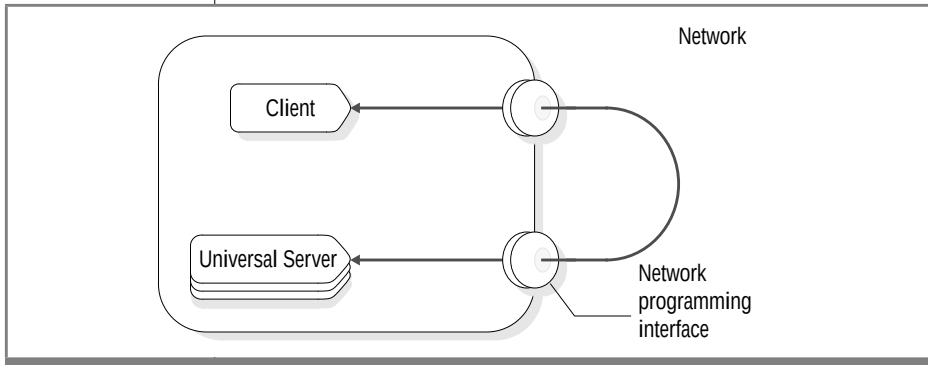


Figure 4-3
*Conceptual
Illustration of Local
Loopback
Connection*

You can think of a local loopback connection as shown in Figure 4-3, but the diagram in Figure 4-4 is a more accurate representation of local loopback. Figure 4-4, which illustrates a local loopback connection, differs from the shared-memory diagram in [Figure 4-1 on page 4-12](#) only in the type of connection between the client application and the database server.

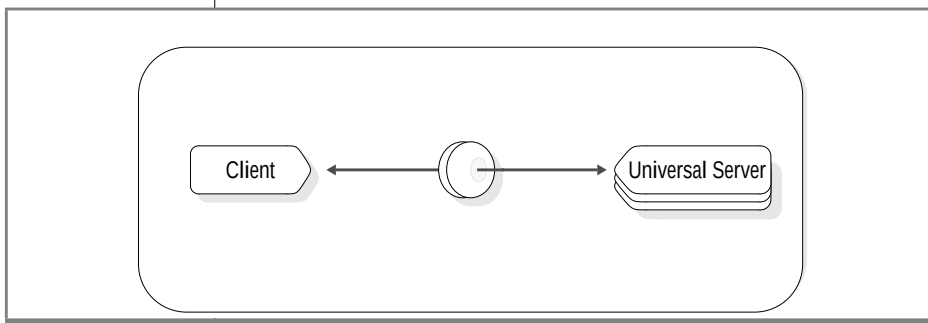


Figure 4-4
*A Simple Local
Loopback
Connection*

PC Connections

This section explains how to configure Universal Server so that you can connect client applications that are running on a DOS- or Windows-based personal computer. This section does *not* explain how you configure your client environment to enable PC-to-Universal Server connections. For information on how to configure your client environment, consult the manual that accompanies your client application or the manual that accompanies the tool with which you built the client application (such as INFORMIX-ESQL / C or INFORMIX-4GL).

Which Protocols Does the Database Server Support for PC Communications?

Universal Server supports two network protocols that facilitate the transfer of data between a PC client application and a Universal Server for UNIX database server: TCP / IP and IPX / SPX. For communication between PC client and UNIX server to be successful, the networks on which the PC client application and UNIX database server reside must support the same protocol.

Which Protocol Should You Use: TCP/IP or IPX/SPX?

If all your PC client applications reside on a network that supports TCP / IP, configure Universal Server to use TCP / IP.

If all your PC client applications reside on a network that supports IPX / SPX, configure Universal Server to use IPX / SPX if the following statements are true:

- IPX / SPX software is available for the database-server computer.
- Your Universal Server database server supports the IPX / SPX network protocol.

If IPX / SPX is not available for the database server computer, and you have client applications that must connect to a database server that supports TCP / IP, you must purchase software that supports TCP / IP for each of these client computers.

However, if both TCP/IP and IPX/SPX are available and are supported on the database server side, you might think about configuring Universal Server with two listen threads, one for TCP/IP and one for IPX/SPX. For instructions on how to add additional listen threads, see [“Adding Listen Threads” on page 10-37](#).

Which protocol you use might be a moot question if only one of the two possible protocols is supported on the platform where Universal Server resides. You can determine which protocols or interfaces are supported by checking your machine-notes file, as described in [“What Interface / Protocol Combinations Are Available on Your Platform?” on page 4-6](#).

Configuring the Database Server for PC Client Applications with TCP/IP

Connecting to Universal Server for UNIX using TCP/IP is often the most convenient method of connection for the database server. Because all UNIX operating systems include software that supports the TCP/IP protocol, you need not install any additional communications software on your UNIX host computer.

If you already installed and configured Universal Server for communication with other TCP/IP client applications, you do not need to alter the configuration of Universal Server in any way. Follow the instructions detailed in the manual that accompanies your client application for setting up connections to a Universal Server.

If you recently installed Universal Server and have not configured Universal Server for connection to TCP/IP client applications, take the following steps:

- Set up the connectivity files as explained in [“Connectivity Files” on page 4-18](#).
- Set up the connectivity configuration parameters as explained in [“ONCONFIG Parameters for Connectivity” on page 4-44](#).

Configuring the Database Server for PC Client Applications with IPX/SPX

If you decide to use the IPX/SPX protocol for PC to Universal Server communication, take the following steps:

- Purchase and install IPX/SPX software on the server where Universal Server is installed.
- Set up the connectivity files, as explained in [“IPX/SPX Connectivity Files” on page 4-21](#).
- Set up the connectivity configuration parameters, as explained in [“ONCONFIG Parameters for Connectivity” on page 4-44](#).

Supplying Configuration Information to Client-Side Administrators

For both the TCP/IP and IPX/SPX protocols, you might need to supply the following configuration information to the person who configures the client environment for communication with Universal Server:

- Host name where Universal Server is running
- Service name
- DBSERVERNAME

Information on the **hostname** and **servicename** fields is included in the following section; information about DBSERVERNAME is included in [“ONCONFIG Parameters for Connectivity” on page 4-44](#).

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, as in data replication or distributed joins. The connectivity configuration files can be divided into three groups:

- Network-configuration files
- Network-security files
- **\$INFORMIXDIR/etc/sqlhosts**

The following sections describe each of these files. Of these files, the Universal Server administrator manages only the Informix **sqlhosts** file. You must have an **sqlhosts** file on each computer that has either a client application or a database server. The other files are UNIX operating-system files that are managed by the UNIX system (or network) administrator or by the end user.

The following sections describes the connectivity files that must be present for client/server connection to succeed. Because the connectivity files required for each communications protocol might be different, they are discussed in separate sections.

Network-Configuration Files

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

TCP/IP Connectivity Files

When you configure Universal 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. Whenever you add a host, or a software service such as a Universal Server database server, you need to inform the network administrator so that he or she can make sure the information contained 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 computer on the network that uses an Informix client/server product. 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 4-30](#) includes a sample **/etc/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/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 Universal Server:

```
unisrvr2      1526/tcp
```

This entry makes **unisrvr2** known as the service name for TCP port 1526. A Universal Server database server can then use this port to service client-application connection requests. [Figure 4-14 on page 4-37](#) includes a sample **/etc/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 these commands at the system prompt: `yycat hosts` and `yycat services`.

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

IPX/SPX Connectivity Files

To configure Universal 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. Which IPX/SPX software you purchase 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 the IPX/SPX software that you purchase.

Network-Security Files

Informix products follow standard UNIX 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 (that is, entries in **/etc/passwd** and, if appropriate, **/etc/shadow**).

The /etc/hosts.equiv and ~/.rhosts Files

The **/etc/hosts.equiv** and **~/.rhosts** files are optional UNIX files that can be created and stored on the computer that runs the database server. These two files are required for the default authentication policy. They specify which remote hosts and users are trusted by the host computer on which the database server resides. Trusted users are allowed to access the database server computer without supplying a password. The database server uses these files to determine whether a remote client should be allowed access to the server without specifying a password explicitly.

The Universal Server administrator can use the **/etc/hosts.equiv** file located on the database server computer to specify a list of trusted hosts that can log in to the database server computer without a password. Alternatively, individual users can maintain their own **.rhosts** file in their home directory on the database server computer.



For information about the UNIX security procedures and trusted computers, refer to the documentation for your installation and to the UNIX manual pages for **hosts.equiv**.

Warning: *On some networks, the host name used by the network to refer to a computer might not be exactly the same as the host name used by computer to refer to itself. For example, the network host name might contain the full domain name as shown in the following example:*

```
viking.informix.com
```

The computer might refer to itself using the local host name shown in the following example:

```
viking
```

*If this occurs, make sure that you specify both host name formats in your **/etc/host.equiv** and **.rhosts** files.*

The ~/.netrc File

The **.netrc** file is an optional UNIX file in the home directory of the user that specifies user identity data. A user who is not a trusted user or not on a computer that is trusted by the remote database server can use this file. The user can also use the **.netrc** file if the user has a different user name and password on the remote computer.



Important: *The **.netrc** file is used regardless of whether a communications support module (CSM) or the default authentication policy is used.*

If the user does 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.

For information about this file, refer to the documentation for your installation and to the UNIX manual pages for **.netrc**.

The \$INFORMIXDIR/etc/concsm.cfg file

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

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

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

The *csname* 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.

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

This parameter assigns the name **dcecss** 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
dbserver principal name	p	string (no quotes)	<i>dbservername</i>

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:

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

For an overview of the communications support services, refer to [“What Are Communications Support Services?” 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 -13](#). For information on how to specify a communications support module in the **sqlhosts** file, refer to [“The Communications Support Module Option” on page 4-43](#).

The \$INFORMIXDIR/etc/sqlhosts File

The Informix **sqlhosts** file contains information that you supply that lets a client application find and connect to an Informix database server anywhere on the network. The **sqlhosts** file must contain an entry (one line) that you supply for each type of connection to each database server on the network. Each entry in the **sqlhosts** file has five fields, as follows:

- The **dbservername** field
- The **nettype** field
- The **hostname** field
- The **servicename** field
- The **options** field

The first four fields are required. The **options** field is optional. The fields are covered in detail in the next sections. Figure 4-5 shows a sample **sqlhosts** file.

Figure 4-5
Sample sqlhosts File

dbservername	nettype	hostname	servicename	options
menlo	onipcshm	valley	shm_file	
menlo2	ontlitcp	valley	menlo_on	
newyork	ontlitcp	hill	online2	

(1 of 2)

dbservername	nettype	hostname	servicename	options
pittsburgh	onsoctcp	canyon	1536	
sales	ontlisp	knight	sales	k=0,r=0
payroll	onsoctcp	dewar	py1	s=2,b=5120
personnel	ontlitcp	37.1.183.92	sales_ol	
texas_Usrvr	ontlitcp	*texas	pd1_on	

(2 of 2)

You can set the **INFORMIXSQLHOSTS** environment variable to the full pathname and filename of a file that contains the **sqlhosts** file information. The content of the specified file must include the same information and have the same format as specified for the **\$INFORMIXDIR/etc/sqlhosts** file. If you do not set **INFORMIXSQLHOSTS**, the database server looks for the **\$INFORMIXDIR/etc/sqlhosts** file.

A client application uses the **sqlhosts** file when it issues a connection statement, as the following example shows:

```
CONNECT TO '@dbservername'
```

The **dbservername** corresponds to an entry in the **dbservername** field of the **sqlhosts** file. For example, using the **sqlhosts** file in Figure 4-5, the client application could issue the following statement:

```
CONNECT TO '@menlo2'
```

The entry for **menlo2** provides the information required for the client application to complete a connection to the **menlo2** database server. To connect to the **menlo2** database server and open database **baseinfo**, the client application would issue the following statement:

```
CONNECT TO 'baseinfo@menlo2'
```

The **CONNECT** statement is fully documented in the [Informix Guide to SQL: Syntax](#).

If you install INFORMIX-SE or INFORMIX-Enterprise Gateway *with DRDA* in the same directory as Universal Server, your **sqlhosts** file will also contain entries for the SE, Gateway, and non-Informix database servers. However, this manual covers only the entries for Universal Server. For information about other entries in the **sqlhosts** file, please refer to the *INFORMIX-SE Administrator's Guide* and the *INFORMIX-Enterprise Gateway with DRDA User Manual*.

Editing the sqlhosts File

You can enter the information required in the **sqlhosts** file by using any convenient text editor. For entries that refer to Universal Server, you must observe the following syntax rules:

- The **dbservername** field can include any printable character other than an uppercase character, a field delimiter, a new-line character, or a comment character. It is limited to 18 characters.
- The **nettype** field can include the values summarized in [Figure 4-7 on page 4-29](#).
- The **hostname** and **servicename** fields can include any printable character other than a field delimiter, a new-line character, or a comment character. The **hostname** field is limited to 256 characters, and the **servicename** field is limited to 128 characters.
- The **options** field can include the values discussed in [“The options Field” on page 4-38](#). The maximum length of the options field is 256 characters.

The fields can be delimited by spaces or by tabs. You cannot include any spaces or tabs *within* a field. You can put comments into the **sqlhosts** file by starting a line with the comment character (#).

The dbservername Field

The **dbservername** (database server name) field contains the name of a database server (dbservername), 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 4-44](#). Each database server across all of your associated networks must have a unique dbservername. If the **sqlhosts** file has multiple entries with the same dbservername, only the first one is used.

The database server uses the dbservername as the key to an index to look up the connectivity information in the remaining fields. Universal Server looks up this information whenever you initialize Universal Server and when client applications attempt to connect to a database server.

The nettype Field

The **nettype** field describes the type of interface / protocol combination that should be made between the client application and the database server.

The **nettype** field is a series of eight letters composed of three subfields, as illustrated in Figure 4-6.

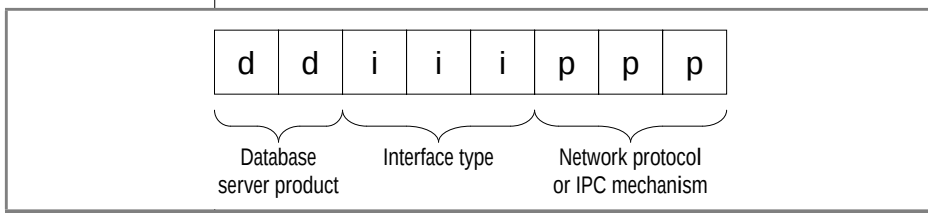


Figure 4-6
*Format of the
nettype Field*

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

The First Two Letters of nettype

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

- on Universal Server (the recommended form) or OnLine Dynamic Server
- ol Universal Server or OnLine Dynamic Server
- se INFORMIX-SE
- dr INFORMIX-Enterprise Gateway with DRDA

The Middle Three Letters of nettype

The middle three letters of **nettype** represent the network programming interface (see [“What Is a Network Programming Interface?” on page 4-5](#)) that enables communications, as follows:

- ipc IPC (interprocess communications)
- soc sockets
- tli TLI (transport layer interface)

Interprocess communications (IPC) are UNIX-based connections used only for communications between two processes running on the same computer. (For information about ipcshm connections, refer to [“The Communications Portion of Shared Memory” on page 12-33](#).) In addition to IPC, Universal Server supports two other network programming interfaces. For more information, see [“Which Network Programming Interfaces Does the Database Server Support?” on page 4-6](#).

The Final Three Letters of nettype

The final three letters of **nettype** represent the specific IPC mechanism or the communications protocol (see [“What Is a Network Protocol?” on page 4-4](#)), as follows:

- shm shared-memory communication
- str stream-pipe communication
- tcp TCP/IP network protocol
- spx IPX/SPX network protocol

IPC connections use shared memory or stream pipes. Universal Server supports two network protocols: TCP/IP and IPX/SPX. For more information on the protocols that Universal Server supports, see [“Which Network Protocols Does the Database Server Support?”](#) on page 4-4.

Figure 4-7 summarizes the **nettype** values for Universal Server. However, Universal Server might not support all these **nettype** values for your platform. See [“What Interface / Protocol Combinations Are Available on Your Platform?”](#) on page 4-6 for information on how you can check which **nettype** values Universal Server supports on your platform.

Figure 4-7
Summary of nettype Values for Universal Server

nettype	Description	Connection Type
onipcshm	Universal Server using shared-memory communication	IPC
onipcstr	Universal Server using stream-pipe communication	IPC
ontlitcp	Universal Server using TLI with TCP/IP protocol	network
onsoctcp	Universal Server using sockets with TCP/IP protocol	network
ontlisp	Universal Server using TLI with IPX/SPX protocol	network

The hostname Field

The third field of the **sqlhosts** file is termed the **hostname** field because in many configurations it contains the name of the computer where the database server resides. The following sections explain how client applications derive the values used in the **hostname** field.

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 hostname of the computer where Universal Server runs.

Network Communication Using TCP/IP

When you use the TCP / IP network protocol, the **hostname** field serves as a key to the **/etc/hosts** file, which gives the network address of the computer. The *hostname* that you use in the **sqlhosts** file must correspond to the *hostname* in the **/etc/hosts** file. Figure 4-8 shows the relationship between the **sqlhosts** file and the **/etc/hosts** file for TCP / IP connections.

Figure 4-8
*Relationship of
sqlhosts File with
the /etc/hosts File
When Using TCP/IP*

\$INFORMIXDIR/etc/sqlhosts file

dbservername	nettype	hostname	servicename	options
menlo2	onsoctcp	valley	menlo_on	
newyork	ontlitcp	hill	unisrvr2	
pittsburgh	onsoctcp	canyon	unisrvr3	

/etc/hosts file

IP address	hostname	host alias(es)
23.33.123.11	hill	sales
4.444.4.44	canyon	acct
333.3.33.33	valley	

Using an IP Address with TCP/IP Connections

For TCP/IP connections (both TLI and sockets), you can now use the actual internet IP address in the host name field instead of the host name or alias found in the **/etc/hosts** file. The IP address is always composed of four sets of one to three integers, separated by periods. Figure 4-9 shows a sample **/etc/hosts** file with IP addresses and host names. The host aliases are optional and can be omitted.

Figure 4-9
A Sample **/etc/hosts** File

Internet IP address	Host name	Host alias(es)
157.11.192.127	smoke	
49.192.4.63	odyssey	
37.1.183.92	knight	sales

Using the IP address from Figure 4-9, the following two **sqlhosts** entries are equivalent.

servername	nettype	hostname	servicename	options
sales	ontlitcp	37.1.183.92	sales_ol	
sales	ontlitcp	knight	sales_ol	

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

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.

Wildcard Addressing with TCP/IP Connections

You can use wildcard addressing in the **hostname** field of the **sqlhosts** file 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 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 4-10 on page 4-33](#), the **/etc/hosts** file must have an entry for each interface card. For example, the **/etc/hosts** file for the **texas** computer might include these entries.

	Internet IP Address	Host Name	Host Alias(es)
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 4-11 on page 4-34](#). The wildcard in the **hostname** field is meaningful only to the database server. If a client application uses an **sqlhosts** file entry that contains a wildcard, the client application simply ignores the wildcard and searches for a host name after the wildcard.

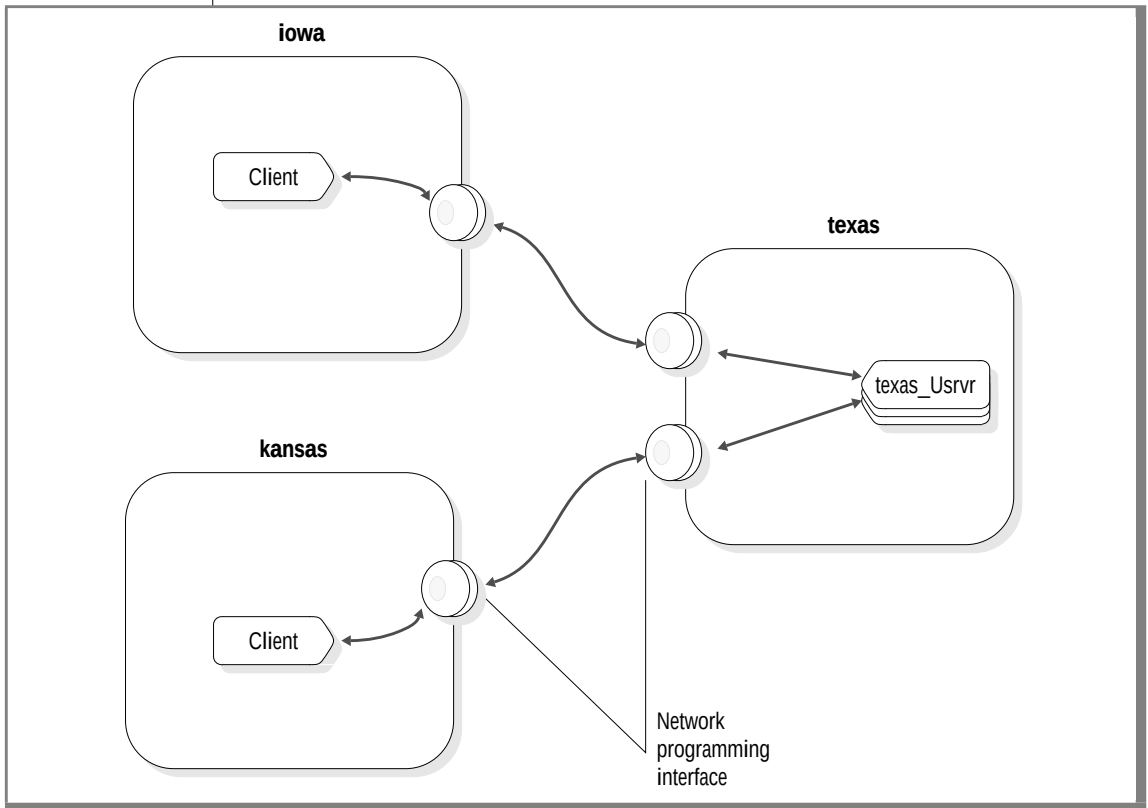
If the client application and database server share an **sqlhosts** file, you can specify both the asterisk and a host name or IP address in the **hostname** field (for example, *texas1 or *123.34.6.81). The client application ignores the asterisk and uses the host name (or IP address) to make the connection, 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 4-10 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 4-10
Using Multiple Network-Interface Cards



The **sqlhosts** file for the **texas_Usrvr** database server can include any one of the entries in Figure 4-11.

Figure 4-11

Possible Entries in the sqlhosts File for the texas_Usrvr Database Server

servername	nettype	hostname	servicename	options
texas_Usrvr	ontlitcp	*texas1	pd1_on	
texas_Usrvr	ontlitcp	*123.34.6.81	pd1_on	
texas_Usrvr	ontlitcp	*texas2	pd1_on	
texas_Usrvr	ontlitcp	*123.34.6.82	pd1_on	
texas_Usrvr	ontlitcp	*	pd1_on	



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

If any of the preceding lines are in its **sqlhosts** file, the **texas_Usrvr** 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 4-12 in its **sqlhosts** file.

Figure 4-12

Possible Entries in the sqlhosts File for the Client Application on iowa

servername	nettype	hostname	servicename	options
texas_Usrvr	ontlitcp	*texas1	pd1_on	
texas_Usrvr	ontlitcp	*123.34.6.81	pd1_on	

(1 of 2)



servername	nettype	hostname	servicename	options
texas_Usrvr	ontliscp	texas1	pd1_on	
texas_Usrvr	ontliscp	123.34.6.81	pd1_on	

(2 of 2)

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 4-13 in its **sqlhosts** file.

Figure 4-13

Possible Entries in the sqlhosts File for the Client Application on kansas

servername	nettype	hostname	servicename	options
texas_Usrvr	ontliscp	*texas2	pd1_on	
texas_Usrvr	ontliscp	*123.34.6.82	pd1_on	
texas_Usrvr	ontliscp	texas2	pd1_on	
texas_Usrvr	ontliscp	123.34.6.82	pd1_on	



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

Network Communication Using 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 required. 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 Field

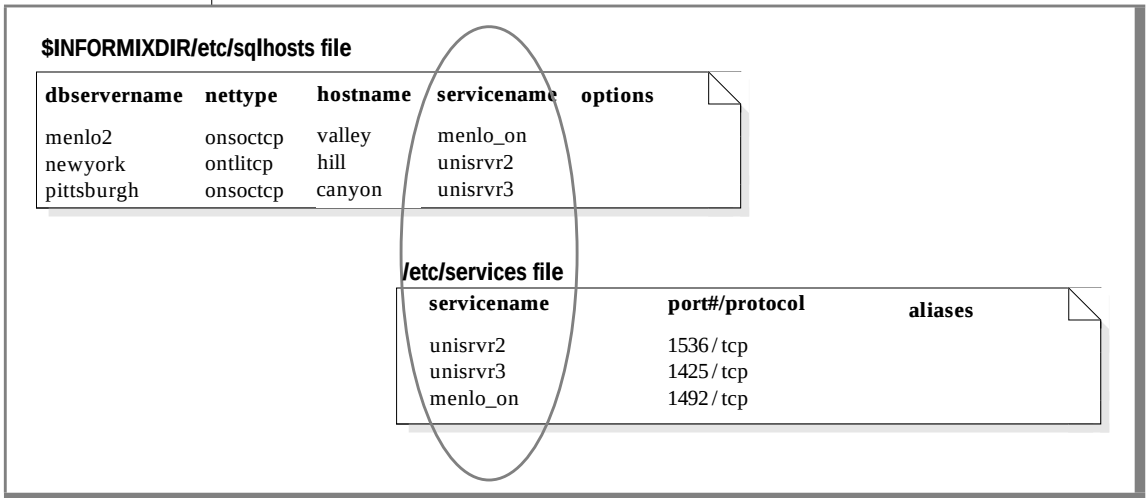
The interpretation of the **servicename** field depends on the type of connection specified in the **nettype** field.

Shared-Memory and Stream-Pipe Communication

When the **nettype** field specifies a shared-memory connection (**onipcshm**) or a stream-pipe connection (**onipcstr**), Universal 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 Universal Server resides. Informix recommends that you use the **dbserv-
ername** as the **servicename** for stream-pipe connections.

Network Communication Using TCP/IP

When you use the TCP/IP connection protocol, the **servicename** must correspond to a **servicename** entry in the **/etc/services** file, as illustrated in [Figure 4-14 on page 4-37](#). The port number in the **/etc/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 4-14*Relationship of sqlhosts File and /etc/services File When Using TCP/IP*

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 **/etc/services** file. You can also use the UNIX **arp** or **ypmatch** command to find the port number. Figure 4-15 shows a sample **/etc/services** file.

Figure 4-15*A Sample /etc/services File*

servicename	port # /protocol	aliases
sales_ol	1536 / tcp	
olport	1425 / tcp	port5

Using the port number from [Figure 4-15 on page 4-37](#), the two **sqlhosts** entries in Figure 4-16 are equivalent.

Figure 4-16
Comparison of Two sqlhosts Entries

servername	nettype	hostname	servicename	options
sales	ontlitcp	knight	sales_ol	
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 less convenient.

Network Communication Using IPX/SPX

A *service* on an IPX/SPX network is simply a program that is prepared to do work for you, such as Universal 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. When you use Version 4.1 of INFORMIX-OnLine *for NetWare*, the service name *must* be the same as the dbservername.

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:

- The keep-alive option
- The security options
- The buffer-size option
- The communications support module option

When you change the options in the **sqlhosts** file, those changes affect the next connection 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. The items must be separated with commas, and no white space is allowed. You can include the items in any order. The following examples show both legal and illegal syntax:

k=0,s=3,b=5120	valid entry
s=3,k=0,b=5120	equivalent to the preceding entry
k = 0,b = 5120	illegal: includes spaces
k=s=0	illegal: cannot combine entries

The **sqlhosts** file in Figure 4-17 shows examples of the keep-alive, security, and buffer-size options that are discussed in the following sections. In line 1, the k=0 in the **options** field disables the keep-alive feature. The r=0 disables ~/.netrc file lookup for the client. In line 2, the s=2 enables /etc/hosts.equiv lookup for the database server, and the b=5120 sets the communications buffer size to 5120 kilobytes. The csm=(dcegss) specifies that the DCE-GSS communications support module is to be used for the connection.

Figure 4-17
A Sample sqlhosts File

	server name	nettype	host name	servicename	options
Line 1	payroll	ontlitcp	dewar	py1	k=0,r=0
Line 2	personnel	ontlisp	skinner	prsnl_ol	s=2,b=5120,csm=(dcegss)

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, 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 Default Authentication Option for the Server

The default authentication option for the server lets you disable the operating-system security-file lookup. The letter *s* identifies server-side settings. You can set this option in the **options** field. A client ignores *s* settings.

The following table shows the possible settings for `s`.

Setting	Result
<code>s=0</code>	Disables both <code>/etc/hosts.equiv</code> and <code>~/rhosts</code> lookup from the server side
<code>s=1</code>	Enables only the <code>/etc/hosts.equiv</code> lookup from the server side
<code>s=2</code>	Enables only the <code>~/rhosts</code> lookup from the server side
<code>s=3</code>	Enables both <code>/etc/hosts.equiv</code> and <code>~/rhosts</code> lookup on the server side (default setting for the server side)

The server default authentication option lets you control the way that a client (user) gains access to a database server by modifying the `sqlhosts` file. By default, an Informix database server searches the `/etc/hosts.equiv` and the `~/rhosts` file of the user to determine whether a client host is trusted. With the security options, you can specifically enable or disable the use of either or both of the `hosts.equiv` and `.rhosts` files.

For example, if you want to prevent end users from specifying trusted hosts in their own `~/rhosts` file, you can disable the `~/rhosts` lookup by setting `s=1` in the `options` field of the `sqlhosts` file for the database server.



Important: Do not disable the `/etc/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`.

The Client Security Option

The letter `r` identifies client-side security settings. Set the `r` option in the `options` field. A client ignores `s` settings, and a database server ignores `r` settings.

The following table shows the possible values for the client security option.

Setting	Result
<code>r=0</code>	Disables the <code>~/netrc</code> lookup from the client side
<code>r=1</code>	Enables the <code>~/netrc</code> lookup from the client side (default setting for the client side)

The client security option applies to both the default authentication policy and to communications support module authentication.

The Buffer-Size Option

Use the buffer-size option (b=) to specify the space (in bytes) reserved for the communications buffer. The buffer-size option applies only to connections that use the TCP/IP network protocol. IPX/SPX, shared-memory, and stream-pipe connections ignore the buffer-size setting.

You can use this option when the default size is not efficient for a particular application. For example, for an application that uses many 32-kilobyte blobs, you could set the size of the communications buffer to 32-kilobytes (assuming that the network server on your operating system supports a 32-kilobyte buffer). The Universal Server default buffer size for TCP/IP is 4096 bytes.

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

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

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.

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, Universal 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 a different CSM option, or no CSM option, for each **sqlhosts** entry.

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

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

The value of *csname* must match a *csname* entry in the **concsbm.cfg** file. The *connection-options* parameter overrides the default *csm-connection* options specified in the **concsbm.cfg** file. Refer to [“The \\$INFORMIXDIR/etc/concsbm.cfg file” on page 4-23](#) for information on the **concsbm.cfg** file entry.

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

```
csm=(dcegss,"p=shm_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 **shm_server**.

For more information on the communications support module, refer to [“What Are Communications Support Services?” on page 4-6](#). For more information on the **concsbm.cfg** file, refer to [“The \\$INFORMIXDIR/etc/concsbm.cfg file” on page 4-23](#).

ONCONFIG Parameters for Connectivity

When you initialize Universal Server, the initialization procedure uses parameter values from the ONCONFIG configuration file. (For a general discussion of Universal Server initialization, refer to [Chapter 9, “What Is Initialization?”](#)) 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 Universal 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 10-31](#).

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 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**, that 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 Universal Server. Figure 4-18 shows entries in an ONCONFIG configuration file that assign three dbservernames to the same Universal Server instance.

Figure 4-18

Example of DBSERVERNAME and DBSERVERALIASES Parameters

DBSERVERNAME	sockets_Usrvr
DBSERVERALIASES	ipx_Usrvr,shm_Usrvr

The **sqlhosts** file associated with the dbservernames from Figure 4-18 could include the entries shown in Figure 4-19. Because each dbservername has a corresponding entry in the **sqlhosts** file, you can associate multiple connection types with one database server.

Figure 4-19

Three Entries in the sqlhosts File for One Database Server

shm_Usrvronipcshmy_hostmy_shm
sockets_Usrvronsoctcpmy_hostport1
ipx_Usrvrontlispnw_file_serveripx_Usrvr

Using the **sqlhosts** file shown in Figure 4-19, a client application uses the following statement to connect to the database server using shared-memory communication:

```
CONNECT TO '@shm_Usrvr'
```

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 Chapter 4 of the [Informix Guide to SQL: Reference](#).

You do not need to set **INFORMIXCONTIME** or **INFORMIXCONRETRY** when you configure and initialize Universal Server. Users of client applications that connect to Universal 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 12-11.

The **INFORMIXSERVER** environment variable allows you to specify a default dbservername to which your clients will connect. For more information, see Chapter 4 of the [Informix Guide to SQL: Reference](#).

Examples of Client/Server Configurations

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

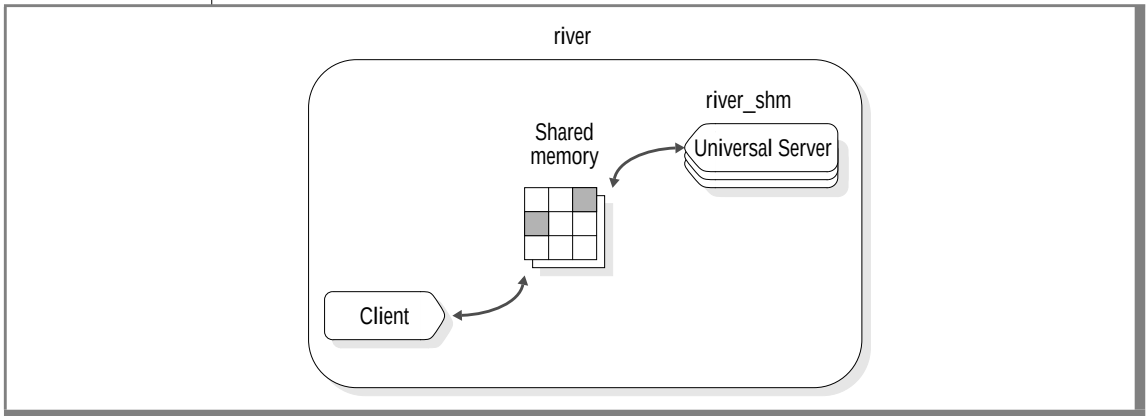
For more information about client / server connections, refer to [“How a Client Attaches to the Communications Portion” on page 12-11](#).

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

Using a Shared-Memory Connection

Figure 4-20 shows a shared-memory connection on the computer named **river**.

Figure 4-20
A Shared-Memory Connection



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

```
DBSERVERNAME river_shm
```

The following shows a correct entry for the **sqlhosts** file:

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.

Using a Local Loopback Connection

Figure 4-21 shows a local loopback connection. The name of the host computer is **river**.

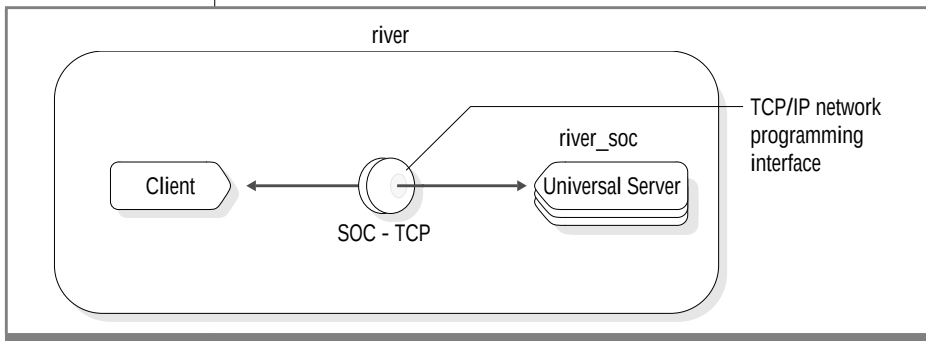


Figure 4-21
Local Loopback Connection

The network connection in Figure 4-21 uses sockets and TCP / IP, so the correct entry for the **sqlhosts** file is as follows:

dbservername	nettype	hostname	servicename
river_soc	onsoctcp	river	riverol

If the network connection uses a TLI interface instead of a sockets interface, only the **nettype** entry in this example changes. In that case, the **nettype** entry is **ontlntcp** instead of **onsoctcp**.

The ONCONFIG file includes the following line:

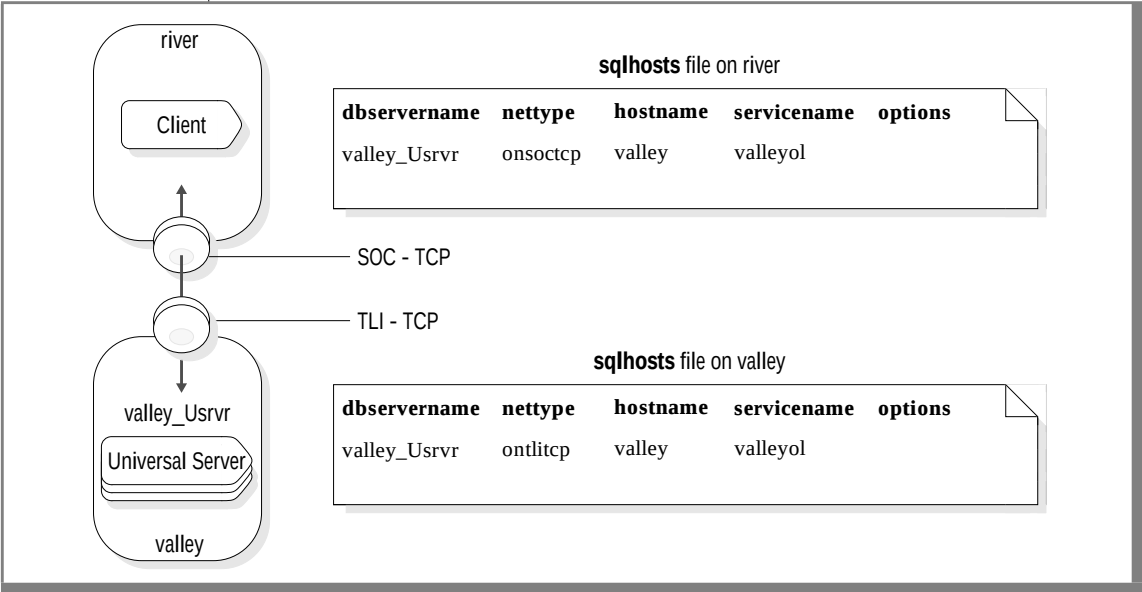
```
DBSERVERNAME river_soc
```

This example assumes that an entry for **river** is in the **/etc/hosts** file and an entry for **riverol** is in the **/etc/services** file.

Using a Network Connection

Figure 4-22 shows a configuration in which the client application resides on host **river** and Universal Server resides on host **valley**.

Figure 4-22
A Network Configuration



An entry for the **valley_Usrvr** database server is in the **sqlhosts** files on both computers. Each entry in the **sqlhosts** file on the computer where the database server resides has a corresponding entry in the **sqlhosts** file of the computer on which 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 the **sqlhosts** file resides. In this example, the **nettype** field for the **valley_Usrvr** database server on host **river** is **onsoctcp**, and the **nettype** field for the **valley_Usrvr** database server on host **valley** is **ontlitycp**.

The sqlhosts File Entry for IPX/SPX

IPX/SPX software frequently provides a TLI interface. If the configuration in [Figure 4-22 on page 4-49](#) uses IPX/SPX instead of a TCP/IP, the entry in the **sqlhosts** file on both computers is as follows:

dbservername	nettype	hostname	servicename
valley_us	ontlisp	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 Universal Server instance can provide more than one type of connection. [Figure 4-23 on page 4-51](#) illustrates such a configuration. Client A connects to the database server using 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.

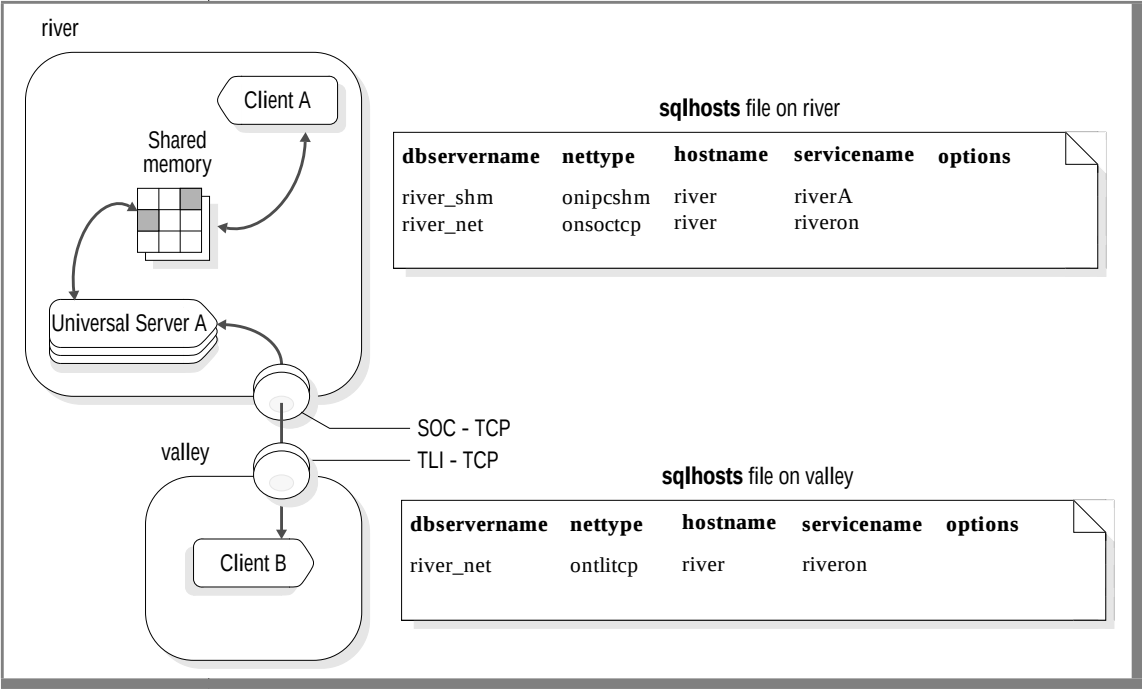
When you want Universal Server to accept more than one type of connection, you must take the following actions:

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

For the configuration in Figure 4-23, the database server has two dbserver-names: **river_net** and **river_shm**. The ONCONFIG configuration file includes the following entries:

```
DBSERVERNAME      river_net
DBSERVERALIASES   river_shm
```

Figure 4-23
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, 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:

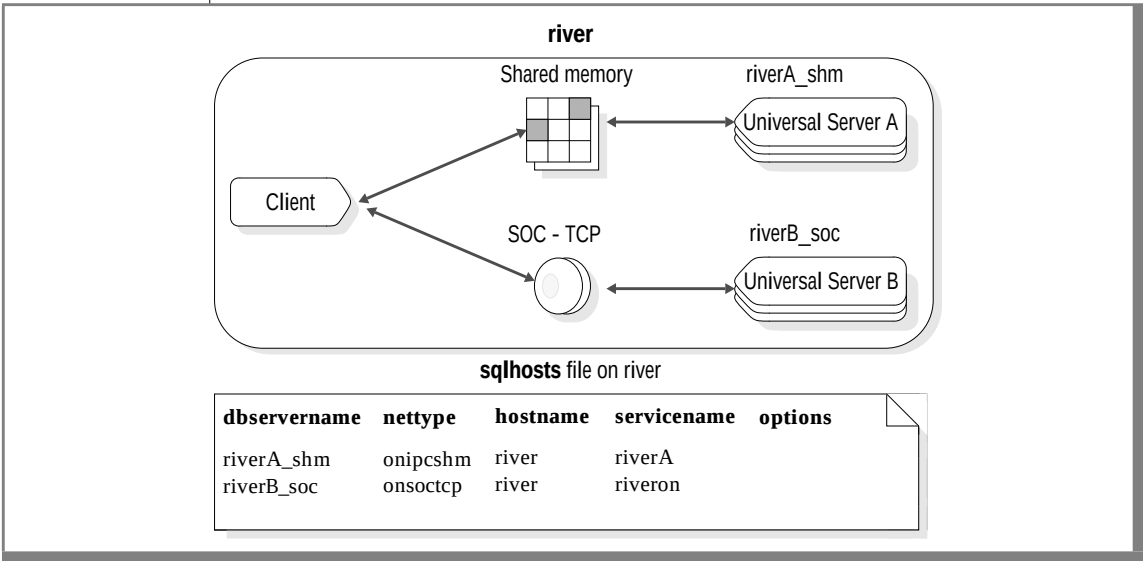
```
CONNECT TO '@river_net'
```

In the **sqlhosts** file, the **nettype** value associated with **river_net** specifies a network (TCP/IP) connection, so client B uses a network connection.

Accessing Multiple Database Servers

Universal Server is on host **river**. When more than one Universal Server is active on one computer, it is known as *multiple residency*. (For information about multiple residency, see [Chapter 5, “What Is Multiple Residency?”](#))

Figure 4-24
Multiple Database Servers



For the configuration in Figure 4-24, you must prepare two ONCONFIG configuration files, one for Universal Server **A** and the other for Universal Server **B**. The **sqlhosts** file includes the connectivity information for both database servers.

The ONCONFIG configuration file for Universal Server **A** includes the following line:

```
DBSERVERNAME      riverA_shm
```

The ONCONFIG configuration file for Universal Server **B** includes the following line:

```
DBSERVERNAME      riverB_soc
```

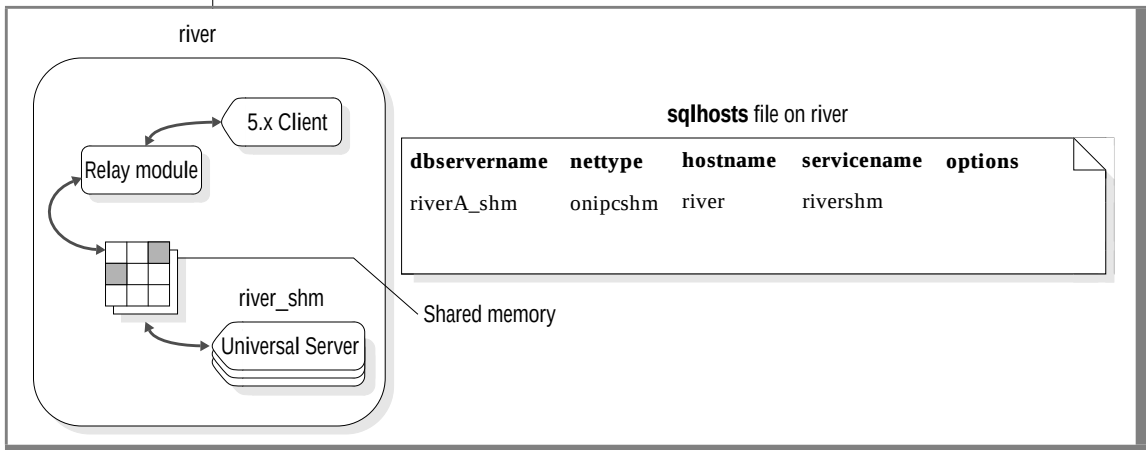
Using the Relay Module

Every Informix database server of Version 6.0 or later includes a *relay module* that lets Version 5.x or Version 4.1 client applications connect to a local database server. The relay module is used when a pre-6.0 client application connects to a post-6.0 database server using a shared-memory connection. For network connections, pre-6.0 client applications can use either pre-6.0 connectivity products such as INFORMIX-NET 5.x, or the relay module. The relay module serves the very important function of allowing connections between Informix products from different release levels. It is designed to be as transparent as possible.

Figure 4-25 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 4-25

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 level works correctly. (In other words, prepare your configuration as you would for the configuration shown in [Figure 4-20 on page 4-47.](#))

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. Universal 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 for some reason you cannot install Universal Server in the same directory as the 5.x Informix products, you must take the following actions:
 - Manually copy or symbolic link the subdirectories from the **\$INFORMIXDIR/msg** directory of the Universal Server product to the **\$INFORMIXDIR/msg** directory of the Version 5.x products.
 - Change the **INFORMIXDIR** environment variable to point to the directory where the Version 5.x products are installed.
 - Modify the **PATH** environment variable to include **\$INFORMIXDIR/bin** of Version 5.x OnLine.
 - 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 Universal Server products are stored. You must use the exact pathname, for example:

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

4. 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
```

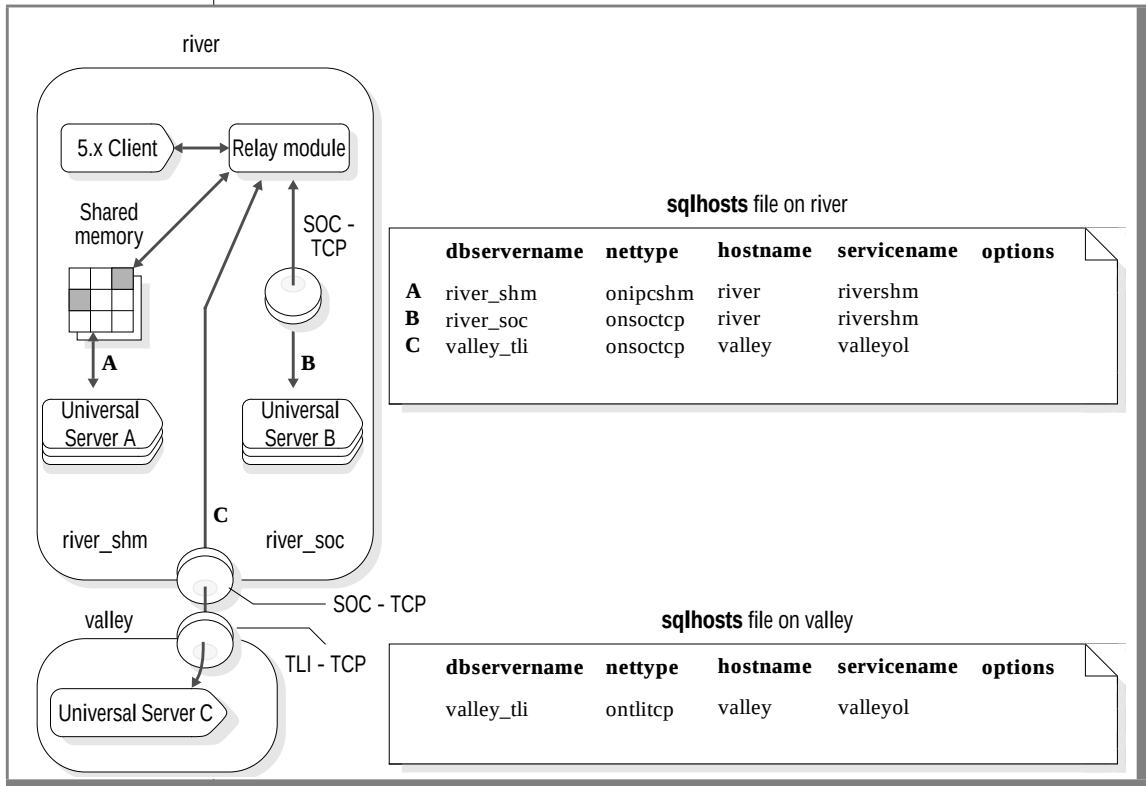
The **DBNETTYPE** environment variable, used by the Version 5.x database servers, is not needed for later database servers. You can unset the **DBNETTYPE** environment variable if you wish, but it does not affect post-6.0 products in any way.

A Relay Module Configuration with Three Database Servers

[Figure 4-26 on page 4-56](#) shows an expanded version of the configuration in [Figure 4-27 on page 4-57](#). This configuration 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 4-26

A Configuration With a 5.x Client, a Relay Module, and Three Database Servers



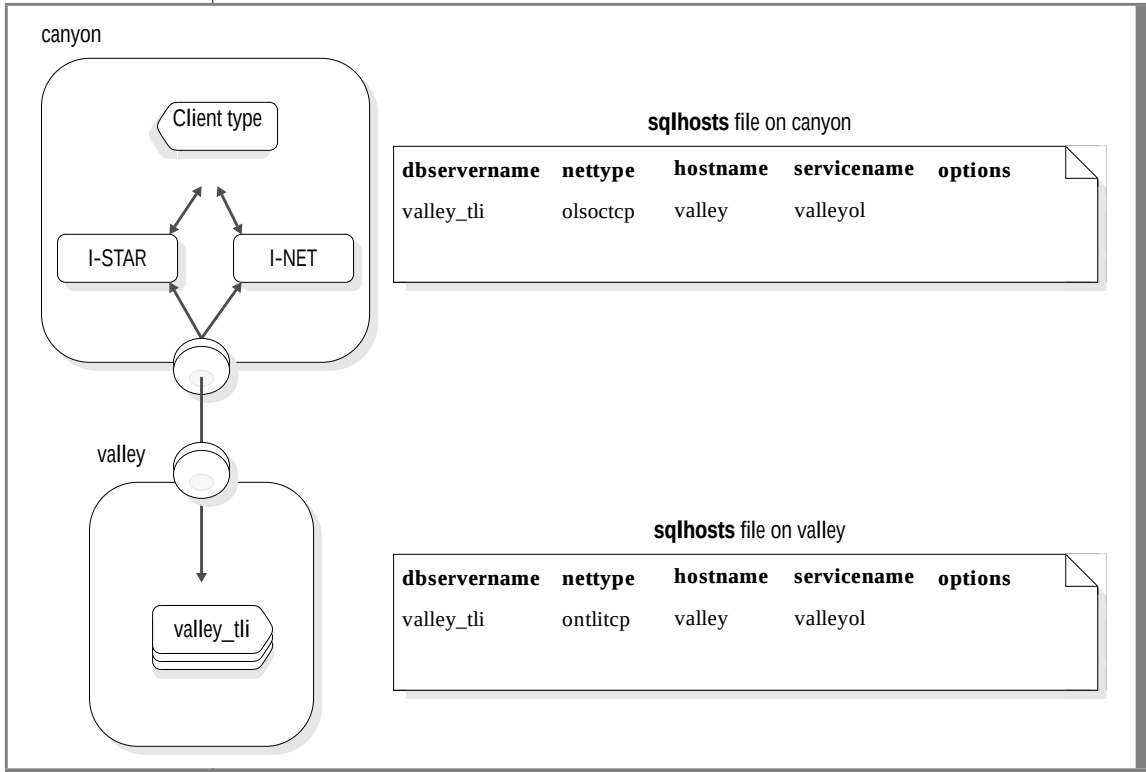
As with [Figure 4-27 on page 4-57](#), verify that all three connections work correctly with a Version 5.x client application, and then modify the environment variables.

Using 5.x INFORMIX-STAR or 5.x INFORMIX-NET

Figure 4-27 illustrates a Version 5.x client application that connects to Universal Server using Version 5.x OnLine with the INFORMIX-STAR client/server product.

Figure 4-27

Example of a Version 5.x Client Application Connected to Universal Server



To use the configuration in Figure 4-27

1. On the server host computer (**valley**), start the database server and verify that a post-6.0 client application can connect to it using a local loopback connection. (For more information, refer to [“Using a Local Loopback Connection” on page 4-48.](#))
2. Write down (for use in step 4) the entry that is in the **sqlhosts** file.

3. On the client host computer (**canyon**), initialize Version 5.x OnLine with INFORMIX-STAR, and verify that it is running correctly. If you are using INFORMIX-NET, no initialization is needed.

For details about starting OnLine with INFORMIX-STAR, refer to the *INFORMIX-NET/INFORMIX-STAR Installation and Configuration Guide*.

4. On the client host, update the **\$INFORMIXDIR/etc/sqlhosts** file associated with INFORMIX-STAR or Version 5.x INFORMIX-NET to contain an entry for Universal Server.

The **sqlhosts** files for the configuration in [Figure 4-27 on page 4-57](#) are the same as they would be if the client were a Version 6.0 client.

If you use DB-Access to test the configuration, you can verify that you are running Version 5.x DB-Access because the first display has only four choices, while the first display of Universal Server DB-Access has six choices. When DB-Access asks for the database, enter the database name and the server name, as in the following example:

```
my_database_name@valley_tli
```

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

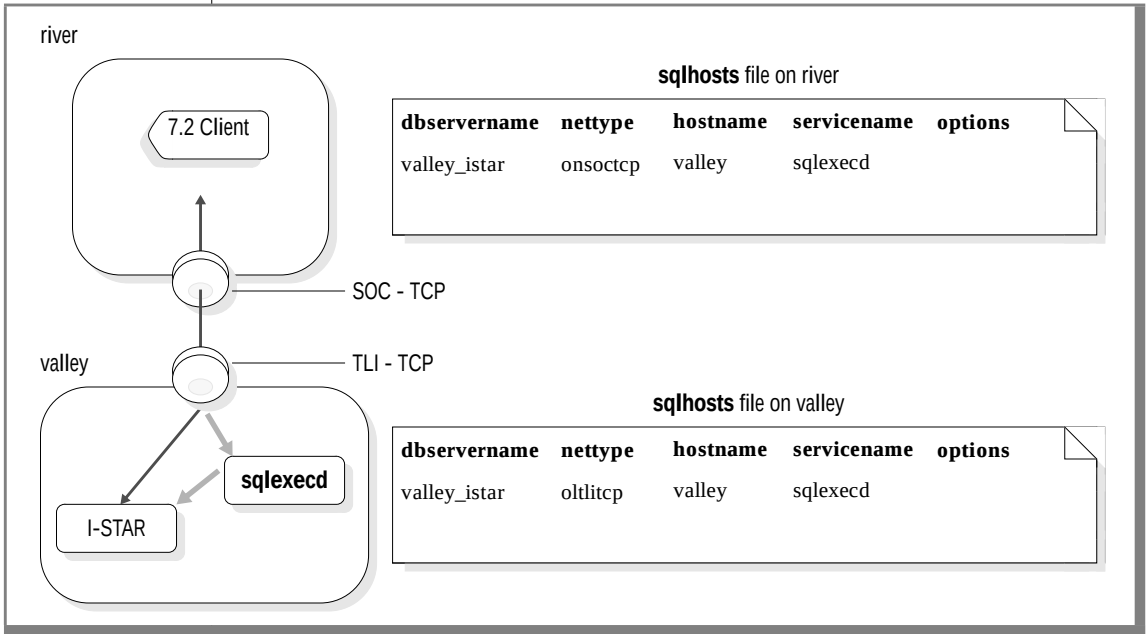
You can connect a post-6.0 client application to Version 5.x OnLine with INFORMIX-STAR using a network connection, but you cannot connect a post-6.0 client application to a Version 5.x OnLine database server using shared memory. You must use a network connection. The post-6.0 client application cannot use any syntax that is specific to later Informix products because the 5.x database server does not recognize later syntax. For example, the post-6.0 client application cannot use the CONNECT statement with a 5.x database server because the CONNECT statement is not supported in Version 5.x.

Instead, the later application must use the 5.x SQL syntax for opening a database server. For example, suppose you want to open a database named **5_base** using the database server **valley_istar**. First, set the environment variable **INFORMIXSERVER** to **valley_istar**. Then, in your application, issue the following statement:

```
DATABASE 5_base@valley_istar
```


Figure 4-28

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



To prepare to use the configuration in Figure 4-28, follow these steps:

1. On the server host computer (**valley**), start Version 5.x OnLine with INFORMIX-STAR, and confirm that it is active by connecting a Version 5.x client application to the database server.
For details about starting OnLine with INFORMIX-STAR, refer to the *INFORMIX-NET/INFORMIX-STAR Installation and Configuration Guide*.
2. Write down (for use in step 5) the entry that is in the **sqlhosts** file for the Version 5.x OnLine with INFORMIX-STAR.
3. On the client host computer (**river**), set the **INFORMIXDIR** environment variable to the directory where the Universal Server products are installed.
4. Modify the **PATH** environment variable to include **\$INFORMIXDIR/bin**.

5. Update the **sqlhosts** file to contain an entry for Version 5.x OnLine with INFORMIX-STAR.
6. Set the **INFORMIXSERVER** environment variable to the dbservername of the Version 5.x OnLine instance with INFORMIX-STAR.

What Is Multiple Residency?

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

Y

ou can use more than one INFORMIX-Universal Server database server in the following two ways:

- By running multiple instances of Universal Server on a single host computer
- By accessing several Universal Server database servers over a network

When multiple Universal Server 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 Universal Server 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 Universal Server 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 4-48](#)) 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 UNIX operating system can maintain separate areas in UNIX shared memory and disk for each instance of Universal Server. Each instance of Universal 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 Universal Server.

The Role of the ONCONFIG Environment Variable

Each instance of Universal 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 Universal Server 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

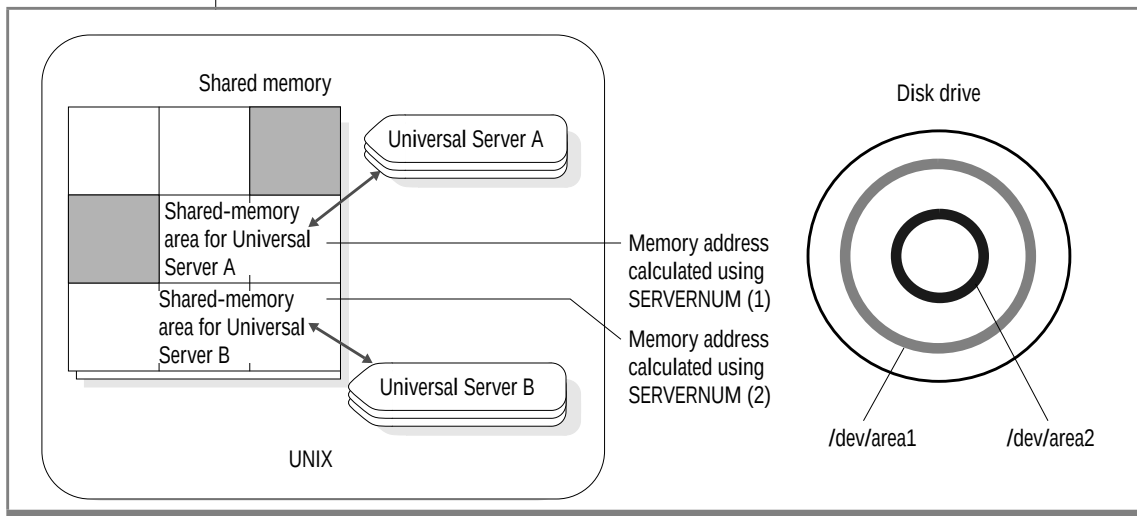
To maintain separation between the instances of Universal Server database servers, you maintain multiple configuration files, each with a unique SERVERNUM value. When you initialize Universal Server, it reads the **ONCONFIG** environment variable for the name of its configuration file. Next, Universal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

For example, the ONCONFIG files for two database servers might include these parameters.

ONCONFIG file: onconfig.one	ONCONFIG file: onconfig.two
...	...
DBSERVERNAME unisrvr_one	DBSERVERNAME unisrvr_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



Using Multiple Residency

Planning for Multiple Residency	6-3
Preparing for Multiple Residency	6-4
Prepare a Configuration File	6-4
Set Your ONCONFIG Environment Variable.	6-5
Edit the New Configuration File	6-5
Add Connection Information	6-6
Update the \$INFORMIXDIR/etc/sqlhosts File	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

T

his chapter describes how to use multiple INFORMIX-Universal Server database servers on the same computer. It includes the following topics:

- Questions that you should ask in planning for multiple residency
- Steps that you should follow for multiple residency

Before you perform this procedure, you should already have installed one Universal Server database server as described in [Chapter 3, “Installing and Configuring.”](#)

Planning for Multiple Residency

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

- Will you use cooked or raw space? Will the raw space share a disk partition with another application?
- 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

Important: Do not try to install another copy of Universal Server binaries. All instances of the same version of Universal Server on one host computer share the same executables.

Here is a summary of the steps for creating another Universal Server on a computer that already has one Universal Server installed:

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 **/etc/services** or connection information to the NetWare server.
5. Update the **\$INFORMIXDIR/etc/sqlhosts** file 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 Universal Server must have its own ONCONFIG configuration file. You prepare an ONCONFIG file for the new instance of Universal Server by copying a configuration file that already exists and modifying 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**.* All configuration files must reside in the **\$INFORMIXDIR/etc** directory.

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 Your ONCONFIG Environment Variable

Set your **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 using 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 this Universal Server configuration. Each instance of Universal 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 this Universal Server database server. Informix suggests that you choose a name that gives information about the database server, such as **ondev37** or **hostnamedev37**. For more information, refer to [“The DBSERVERNAME Configuration Parameter” on page 4-44](#).

- MSGPATH

The MSGPATH parameter specifies the UNIX pathname of the message file for this Universal Server database server. You should specify a unique pathname for the message file because Universal Server messages do not include the dbservername. If multiple Universal Server database servers use the same MSGPATH, you cannot identify the messages from separate Universal 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 Universal Server.

- ROOTPATH and /or ROOTOFFSET

The ROOTPATH and ROOTOFFSET parameters together specify the location of the root dbspace for this Universal Server database server. The root dbspace location must be unique for every Universal 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 37-61](#) and [“ROOTOFFSET” on page 37-61](#).



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 28-3](#).

Add Connection Information

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

Update the \$INFORMIXDIR/etc/sqlhosts File

The **sqlhosts** file must have an entry for each database server. If Informix products on other computers access this instance of Universal Server, the administrators on those computers must update their **sqlhosts** files.

[Chapter 4, “Client / Server Communications,”](#) discusses the preparation of the **sqlhosts** file.

If you plan to use TCP / IP network connections with this instance of Universal Server, the system network administrator must update the **/etc/hosts** and **/etc/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 4-19](#) and [“IPX/SPX Connectivity Files” on page 4-21](#).

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 Universal Server database server, Universal Server initializes the disk space specified in the current ONCONFIG configuration file.



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

Prepare Dbospace and Logical-Log Backup Environment

This section gives a very brief discussion of the effects of multiple residency on backups. If you use ON-Archive for your backup and restore tool, refer to the [INFORMIX-Universal Server Archive and Backup Guide](#) for more information.

When you use multiple residency, you must maintain separate dbospace and logical-log backups for each Universal Server instance. When you perform dbospace 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 Universal Server database server, you can back up your logical-log files using the continuous logging option. Otherwise, you must plan your dbospace and logical-log backup schedules carefully, so that use of a device for one Universal Server instance does not cause the other Universal Server instance to wait. You must reset the ONCONFIG parameter each time you switch your backup operations from one Universal Server instance to the other.

If You Use ON-Archive

If you use ON-Archive for your backup and restore tool, each Universal 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 Universal Server instances before you kill them. The [INFORMIX-Universal Server Archive and Backup Guide](#) explains how to associate the process ID of an **oncatlgr** process with a Universal Server instance.

You can direct Universal Server to use a personalized copy of **config.arc** and **oper_deflt.arc**. You do this by setting 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 Universal Server to use these files as your configuration files, set **ARC_CONFIG** to **config.mine** and **ARC_DEFAULT** to **oper_deflt.mine**.

Update the Operating-System Boot File

You can ask your system administrator to modify the system startup script ([“Prepare UNIX Startup and Shutdown Scripts” on page 3-29](#)) so that each of your Universal Server instances starts whenever the computer is rebooted (for example, after a power failure).

The startup script for a single Universal Server instance should set the **INFORMIXDIR**, **PATH**, **ONCONFIG**, and **INFORMIXSERVER** environment variables and then execute **oninit**. To start a second instance of Universal Server, change the **ONCONFIG** and **INFORMIXSERVER** environment variables to point to the configuration file for the second Universal 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 Universal Server shut down in a graceful manner.

Check Users’ INFORMIXSERVER Environment Variables

If this new instance of Universal Server 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. Chapter 4 of the [Informix Guide to SQL: Reference](#) describes the **informix.rc** and **.informix** files.

Modes and Initialization

Section III



What Are the Operating Modes?

Off-Line Mode	7-3
Quiescent Mode	7-4
On-Line Mode	7-4
Read-Only Mode	7-4
Recovery Mode	7-4
Shutdown Mode.	7-5

T

his chapter explains the operating modes of INFORMIX-Universal Server.

You can determine the current Universal Server mode by executing **onstat**. The mode is displayed in the header. The mode also appears in the status line displayed in ON-Monitor. Universal Server has six modes of operation, as follows:

- Off-line mode
- Quiescent mode
- On-line mode
- Read-only mode
- Recovery mode
- Shutdown mode

For instructions on how to change Universal Server modes, see [Chapter 8, “Managing Modes.”](#)

Off-Line Mode

When Universal Server is in off-line mode, it is not running.

Quiescent Mode

Administrative procedures that require a pause in database activity are performed when Universal Server is in quiescent mode. Only user **informix** or user **root** can access the administrative options of ON-Monitor or perform command-line administrative actions.

In quiescent mode, users cannot connect to a database, but any user can use ON-Monitor or **onstat** to see status information.

On-Line Mode

When Universal Server is in on-line mode, users can connect with the database server and perform all database activities. Universal Server administrator (user **informix** or user **root**) can use the command-line utilities to change many Universal Server ONCONFIG parameter values while Universal Server is on-line.

Read-Only Mode

Read-only mode is used by the secondary database server in a 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

Recovery mode is transitory. It occurs when Universal Server is moving from off-line to quiescent mode. *Fast recovery* is performed when Universal Server is in recovery mode.

(A mirrored chunk can be in recovery *state*, but this state is not the same as Universal Server recovery *mode*.)

Shutdown Mode

Shutdown mode is transitory. It occurs when Universal Server is moving from on-line to quiescent mode or from on-line (or quiescent) to off-line mode. Once shutdown mode is initiated, it cannot be cancelled.

Managing Modes

Users Permitted to Change Modes	8-3
From Off-Line to Quiescent	8-3
How to Perform This Change Using ON-Monitor	8-4
How to Perform This Change Using oninit	8-4
From Off-Line to On-Line	8-4
How to Perform This Change Using oninit	8-4
From Quiescent to On-Line	8-5
How to Perform This Change Using ON-Monitor	8-5
How to Perform This Change Using onmode	8-5
Gracefully from On-Line to Quiescent	8-5
How to Perform This Change Using ON-Monitor	8-6
How to Perform This Change Using onmode	8-6
Immediately from On-Line to Quiescent	8-6
How to Perform This Change Using ON-Monitor	8-7
How to Perform This Change Using onmode	8-7
From Any Mode Immediately to Off-Line	8-7
How to Perform This Change Using ON-Monitor	8-8
How to Perform This Change Using onmode	8-8

T

his chapter contains instructions on changing INFORMIX-Universal Server modes. It describes the following mode changes:

- Off-line to quiescent
- Off-line to on-line
- Quiescent to on-line
- On-line to quiescent (gracefully)
- On-line to quiescent (immediately)
- Any mode to off-line (immediately)

For a description of the modes, see [Chapter 7, “What Are the Operating Modes?”](#)

Users Permitted to Change Modes

Only those users who are logged in as either **root** or **informix** can perform Universal Server mode changes.

From Off-Line to Quiescent

When Universal Server changes from off-line mode to quiescent mode, Universal Server initializes shared memory.

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

How to Perform This Change Using ON-Monitor

To take Universal Server to quiescent mode using ON-Monitor, select the Mode menu, Startup option.

How to Perform This Change Using oninit

Execute **oninit -s** from the command line to take Universal Server from off-line mode to quiescent mode.

To verify that Universal Server is running, execute **onstat** from the command line. The header on the **onstat** output gives the current operating mode.

From Off-Line to On-Line

When you take Universal Server from off-line mode to on-line mode, Universal Server initializes shared memory. You cannot go directly from off-line mode to on-line mode using ON-Monitor.

When Universal Server is in on-line mode, it is accessible to all Universal Server sessions.

How to Perform This Change Using oninit

Execute **oninit** from the command line to take Universal Server from off-line mode to on-line mode.

To verify that Universal Server is running, execute **onstat** from the command line. The header on the **onstat** output gives the current operating mode.

From Quiescent to On-Line

When you take Universal Server from quiescent mode to on-line mode, all sessions gain access.

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

How to Perform This Change Using ON-Monitor

To take Universal Server from quiescent mode to on-line mode using ON-Monitor, select the Mode menu, On-Line option.

How to Perform This Change Using onmode

Execute **onmode -m** to take Universal Server from quiescent mode to on-line mode.

To verify that Universal Server is running in on-line mode, execute **onstat** from the command line. The header on the **onstat** output gives the current operating mode.

Gracefully from On-Line to Quiescent

Take Universal Server gracefully from on-line mode to quiescent mode to restrict access to Universal Server without interrupting current processing.

After you perform this task, Universal Server sets a flag that prevents new sessions from gaining access to Universal 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, Universal Server is considered to be in Shutdown mode.

How to Perform This Change Using ON-Monitor

To take Universal Server from on-line mode to quiescent mode gracefully using ON-Monitor, select the Mode menu, Graceful-Shutdown option.

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.

How to Perform This Change Using onmode

Execute the **onmode -s** or **onmode -sy** options from the command line to take Universal Server gracefully from on-line mode to quiescent mode.

To verify that Universal Server is running in quiescent mode, execute **onstat** from the command line. The header on the **onstat** output gives the current operating mode.

Immediately from On-Line to Quiescent

Take Universal Server immediately from on-line mode to quiescent mode to restrict access to Universal Server as soon as possible. Work in progress can be lost.

A prompt asks for confirmation of the immediate shutdown. If you confirm, Universal 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 automatically comply within 10 seconds, Universal Server terminates this session.

Universal Server users receive either error message -459 indicating that Universal Server was shut down or error message -457 indicating that their session was unexpectedly terminated.

Universal Server performs proper cleanup on behalf of all sessions that were terminated by Universal Server. Active transactions are rolled back.

How to Perform This Change Using ON-Monitor

To take Universal Server immediately from on-line mode to quiescent mode using ON-Monitor, select the Mode menu, Immediate-Shutdown option.

How to Perform This Change Using onmode

Execute **onmode -u** or **onmode -uy** from the command line to take Universal Server immediately from on-line mode to quiescent mode.

To verify that Universal Server is running in quiescent mode, execute **onstat** from the command line. The header on the **onstat** output gives the current operating mode.

From Any Mode Immediately to Off-Line

Take Universal Server immediately from any mode to off-line mode if Universal Server is no longer running. After you take Universal Server to off-line mode, reinitialize shared memory by taking Universal Server to quiescent or on-line mode. When you reinitialize shared memory, Universal Server performs a fast recovery to ensure that the data is logically consistent.

A prompt asks for confirmation to go off-line. If you confirm, Universal 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, Universal Server terminates this session.

Universal Server users receive either error message -459 indicating that Universal Server was shut down or error message -457 indicating that their session was unexpectedly terminated.

Universal Server performs proper cleanup on behalf of all sessions that were terminated by Universal Server. Active transactions are rolled back.

How to Perform This Change Using ON-Monitor

To take Universal Server immediately from any mode to off-line mode, select the Mode menu, Take-Offline option.

How to Perform This Change Using onmode

Execute the **onmode -k** or **onmode -ky** options from the command line to take Universal Server off-line immediately.

A prompt asks for confirmation of the immediate shutdown. The **-y** option to **onmode** eliminates this prompt.

What Is Initialization?

Types of Initialization	9-3
Initialization Commands	9-4
Initialization Steps	9-4
Process Configuration File	9-5
Create Shared-Memory Portions	9-6
Initialize Shared-Memory Structures	9-7
Initialize Disk Space	9-8
Start All Required Virtual Processors	9-8
Make Necessary Conversions	9-8
Initiate Fast Recovery.	9-8
Initiate a Checkpoint	9-9
Document Configuration Changes	9-9
Create the oncfg_servername.servernum File	9-9
Drop Temporary Tblspaces.	9-9
Set Forced Residency If Specified	9-10
Return Control to User	9-10
Prepare SMI Tables	9-10
After Initialization	9-11

Initialization of INFORMIX-Universal 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 happen during initialization.

Types of Initialization

Shared-memory initialization establishes the contents of Universal 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 Universal Server. Universal Server must be in off-line mode when you begin initialization. (For more information about modes, refer to [Chapter 7, “What Are the Operating Modes?”](#))

You can initialize shared memory and disk space using either of the following utilities:

- The **oninit** utility (For more information, refer to [“oninit: Initialize Universal Server” on page 39-27.](#))
- The ON-Monitor utility (For more information, refer to [“Using ON-Monitor” on page 36-3.](#))

The options that you include in the **oninit** command or 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 happen 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 9-1 show the main tasks completed during the two types of initialization. Each step is discussed in the following sections.

Figure 9-1
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.

(1 of 2)

Shared-Memory Initialization	Disk Initialization
	Initialize disk space.
Start all required virtual processors.	Start all required virtual processors.
Make necessary conversions.	
Initiate fast recovery.	
Initiate a checkpoint.	Initiate a checkpoint.
Document configuration changes.	
Update oncfg_servername.servernum file.	Update oncfg_servername.servernum file.
Change to quiescent mode.	Change to quiescent mode.
Drop temporary tablespaces (optional).	
Set forced residency, if requested.	Set forced residency, if specified.
Change to online mode and return control to user.	Change to online mode and return control to user.
If the SMI tables are not current, update the tables.	Create SMI tables.

(2 of 2)

Process Configuration File

Universal 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, Universal Server reads values from the file specified by **\$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 file **\$INFORMIXDIR/etc/onconfig**.
3. If the database server cannot find the **onconfig** file, it reads the configuration values from **\$INFORMIXDIR/etc/onconfig.std**.

Informix recommends that you *always* set the **ONCONFIG** environment variable before you initialize Universal Server. The default configuration files are intended as templates and not as functional configurations. For more information about the configuration file, refer to [“Prepare the ONCONFIG Configuration File” on page 3-14](#).

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 42-8](#). Where differences exist, Universal Server uses the values from the current **ONCONFIG** configuration file for initialization.

Create Shared-Memory Portions

Next, Universal 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, tablespaces, and so on
- Virtual portion, used to track specific tasks for individual users
- IPC communication portion, used for IPC communication. (The database server allocates this portion of shared memory only if you configure an IPC shared-memory connection.)

Next, Universal 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 Database Server Shared Memory” on page 12-28](#).

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 Universal Server Operation” on page 39-75](#).

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. Universal 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 42-6](#).

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 10-5](#).

Make Necessary Conversions

The database server checks its internal files. If the files are from an earlier version of Universal Server, it updates these files to the current format. For information about database conversion, refer to the [Informix Migration Guide](#).

Initiate Fast Recovery

This task is not performed during disk-space initialization because there is not yet anything to recover.

Universal Server checks if fast recovery is needed. If fast recovery is required, the database server initiates fast recovery. For more information about fast recovery, refer to [“What Is Fast Recovery?” on page 26-3](#).

Initiate a Checkpoint

After fast recovery executes, Universal 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 12-56](#).

The database server now moves to quiescent mode or on-line mode, depending on how you started the initialization process.

Document Configuration Changes

This task is not performed during disk-space initialization.

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.

Create the oncfg_servername.servernum File

Universal Server creates the **oncfg_servername.servernum** file and updates it every time 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. 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-10](#).

Drop Temporary Tblspaces

This task is not performed during disk-space initialization.

Universal 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 14-28](#).

Set Forced Residency If Specified

If the value of the `RESIDENT` configuration parameter is 1, the database server tries to enforce residency of shared memory. If the host UNIX 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 37-60](#).

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 37-46](#).

At this point, control returns to the user. Any error messages generated by the initialization procedure are displayed, either at the UNIX command line, within ON-Monitor, or in the message log.

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 38, “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 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 using **onmode** or ON-Monitor or until the system crashes. Informix recommends that you *do not* try to stop the database server by killing a virtual processor or an **oninit** process. For more information, refer to [“Starting and Stopping Virtual Processors” on page 11-7](#).

Disk, Memory, and Process Management

Section IV



What Is the Universal Server Architecture?

What Is a Virtual Processor?	10-5
What Is a Thread?	10-6
What Is a User Thread?	10-6
Types of Virtual Processors	10-7
Advantages of Virtual Processors	10-8
Sharing Processing	10-8
Saving Memory and Resources	10-9
Processing in Parallel	10-10
Adding and Dropping Virtual Processors in On-Line Mode	10-11
Binding Virtual Processors to CPUs	10-11
How Virtual Processors Service Threads	10-12
Control Structures	10-12
Context Switching	10-13
Stacks	10-15
Queues	10-16
Ready Queues	10-16
Sleep Queues	10-16
Wait Queues	10-18
Mutexes	10-18
Virtual-Processor Classes	10-19
CPU Virtual Processors	10-19
How Many CPU Virtual Processors Do You Need?	10-19
Running on a Multiprocessor Computer	10-20
Running on a Single-Processor Computer	10-20
Adding and Dropping CPU Virtual Processors in On-Line Mode	10-21
Preventing Priority Aging	10-21
Using Processor Affinity	10-22

User-Defined Classes of Virtual Processors	10-24
The Purpose of User-Defined Virtual Processors	10-24
Specifying a Nonyielding User-Defined Virtual Processor	10-25
Assigning a User-Defined Routine to a User-Defined Virtual Processor Class	10-25
Adding and Dropping User-Defined Virtual Processors in On-Line Mode	10-26
Disk I/O Virtual Processors	10-26
I/O Priorities	10-27
Logical-Log I/O	10-27
Physical-Log I/O	10-28
Asynchronous I/O	10-28
Network Virtual Processors.	10-31
Specifying Network Connections	10-32
Should Poll Threads Run on CPU or Network Virtual Processors?	10-32
How Many Networking Virtual Processors Do You Need? . . .	10-33
Listen and Poll Threads for the Client/Server Connection . . .	10-33
Starting Multiple Listen Threads	10-37
Optical Virtual Processor	10-39
Audit Virtual Processor	10-39
Miscellaneous Virtual Processor	10-39

INFORMIX-Universal Server embodies an object-relational database-server architecture that has two central principles:

- Extensibility
- Dynamic Scalability

Extensibility means that you can extend the native capabilities of the database server by adding new data types and, if necessary, the supporting routines and access methods to support them. You can add data types and routines using the Universal Server facilities for these tasks, or you can purchase prepackaged data types, including the supporting routines, as DataBlade modules. DataBlade modules are sold by Informix and by third-party vendors. For a more complete description of extensibility, refer to [“What Is Universal Server?” on page 1-3](#).

Scalability refers to the *dynamic scalable architecture* (DSA), or the capability of Universal Server to scale its resources in relation to the demands that applications place on it. DSA provides the following performance advantages, which this chapter describes further, for both single-processor and multiprocessor platforms:

- 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.

Universal 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 Universal Server provides on these systems:

- Multiple processes can work in parallel for one client.
- On some multiprocessor computers, you can bind Universal Server processes to specific CPUs.

Figure 10-1 provides a simplified view of Universal Server, illustrating extensibility through DataBlade modules.

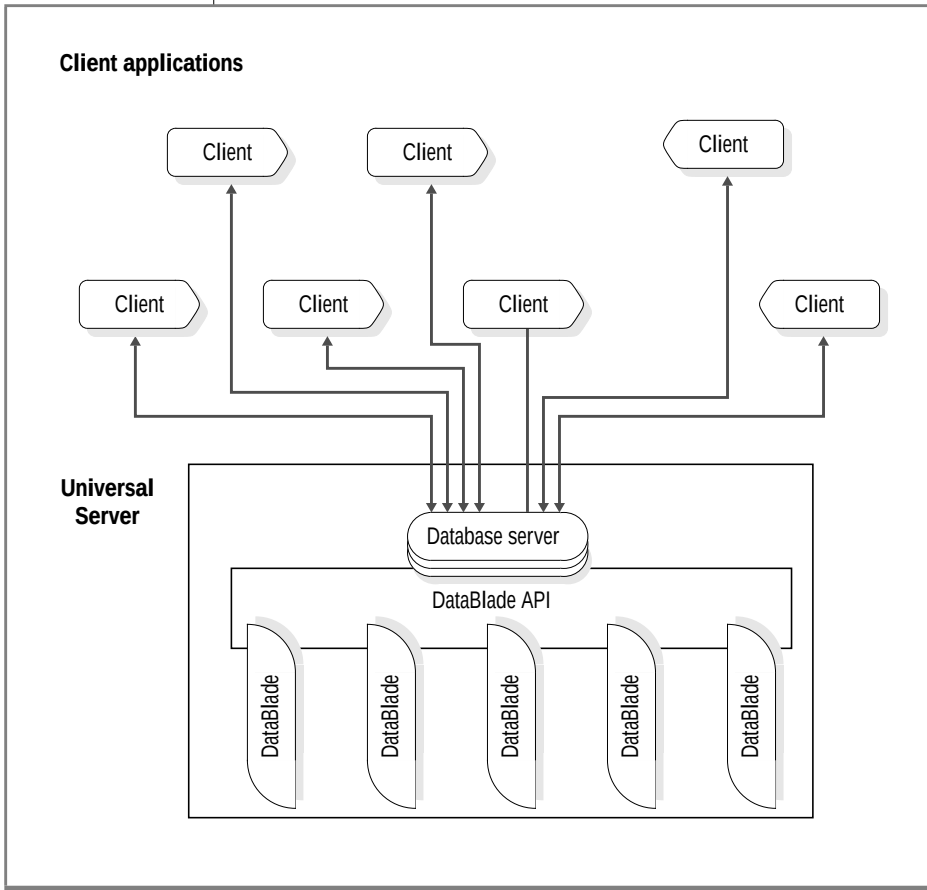


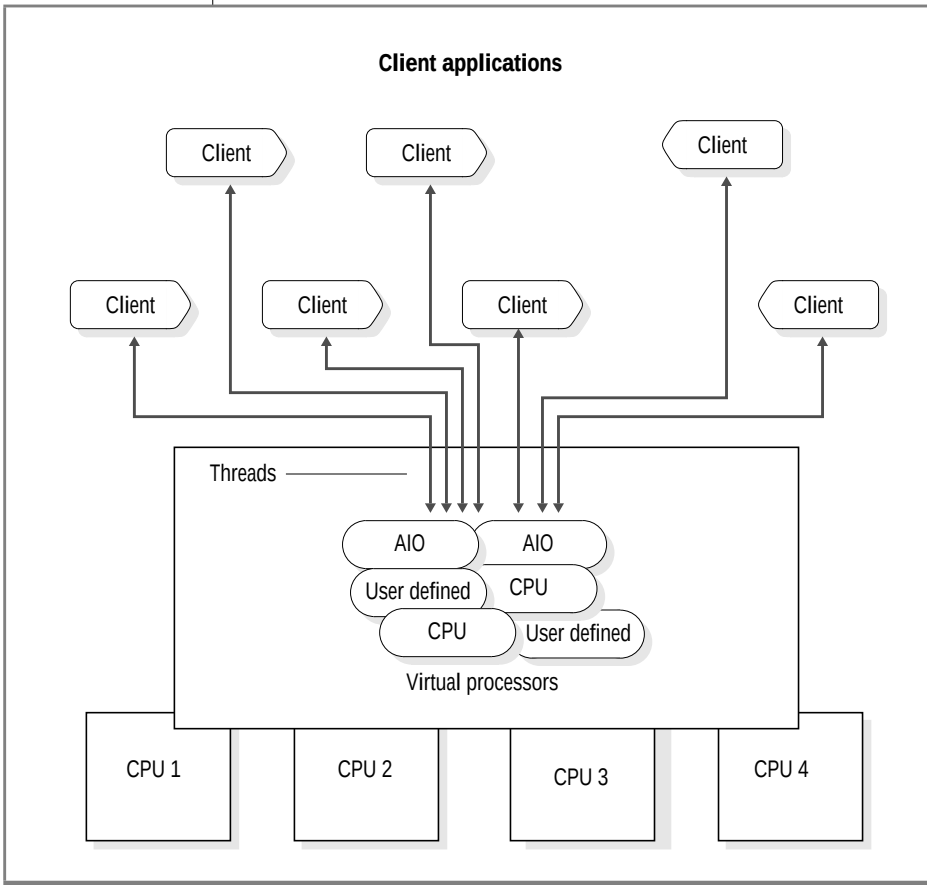
Figure 10-1
*Universal Server
with DataBlade
Module Extensions*

What Is a Virtual Processor?

The central component of the dynamic scalable architecture is the *virtual processor*, which is a Universal Server process. Universal 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.

Figure 10-2 illustrates the relationship of client applications to virtual processors. A small number of virtual processors serve a much larger number of client applications.

Figure 10-2
Dynamic Scalable
Architecture and
Virtual Processors



For information on how to configure virtual processors, refer to [Chapter 11, “Managing Virtual Processors.”](#)

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. As a process, the virtual processor is a task that the operating system schedules for execution on the CPU; a thread is a task that the virtual processor schedules internally for processing. Threads are sometimes called *lightweight processes* because they are like processes, but they make fewer demands on the operating system.

Virtual processors are *multithreaded processes* because they run multiple concurrent threads (MCT).

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, Universal 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 Universal Server runs multiple session threads for a single client, refer to [“Processing in Parallel” on page 10-10](#).

What Is a User Thread?

A *user thread* is a Universal Server thread that services requests from client applications. User threads include session threads, called **sqlxec** 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 33-35](#).

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 10-3 shows the classes of virtual processors and the types of processing that they do.

Figure 10-3
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.

(1 of 2)

Virtual Processor Class	Category	Purpose
ADT	Auditing	Performs auditing functions.
MSC	Miscellaneous	Serves requests for system calls that require a very large stack.
<i>classname</i>	User-defined	Runs user-defined routines in a thread-safe manner so that if the routine fails, the database server is unaffected. Specified using the VPCLASS configuration parameter. You must specify <i>classname</i> .

(2 of 2)

Advantages of Virtual Processors

Compared to a database server process that services a single client application, the dynamic, multithreaded nature of a Universal 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 Universal 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

Universal 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 10-13](#).

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 10-4 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.

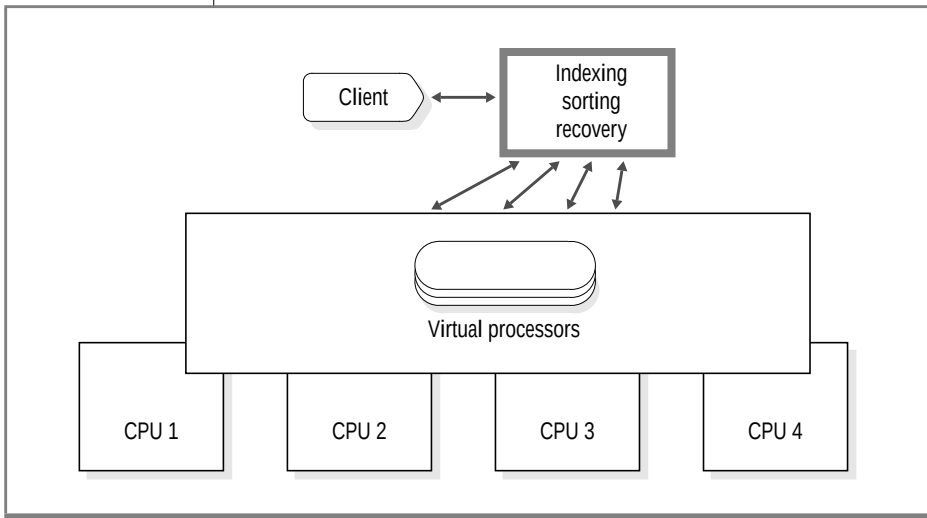


Figure 10-4
Parallel Processing

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 Universal Server is running, you can add virtual processors for any of the classes, but you can only drop virtual processors for the CPU and user-defined classes. 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 11-8](#) and [“Dropping CPU or User-Defined Virtual Processors in On-Line Mode” on page 11-10](#).

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 10-22](#).

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 Universal 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*. Universal 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 Universal 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 10-5 illustrates how a virtual processor accomplishes a context switch.

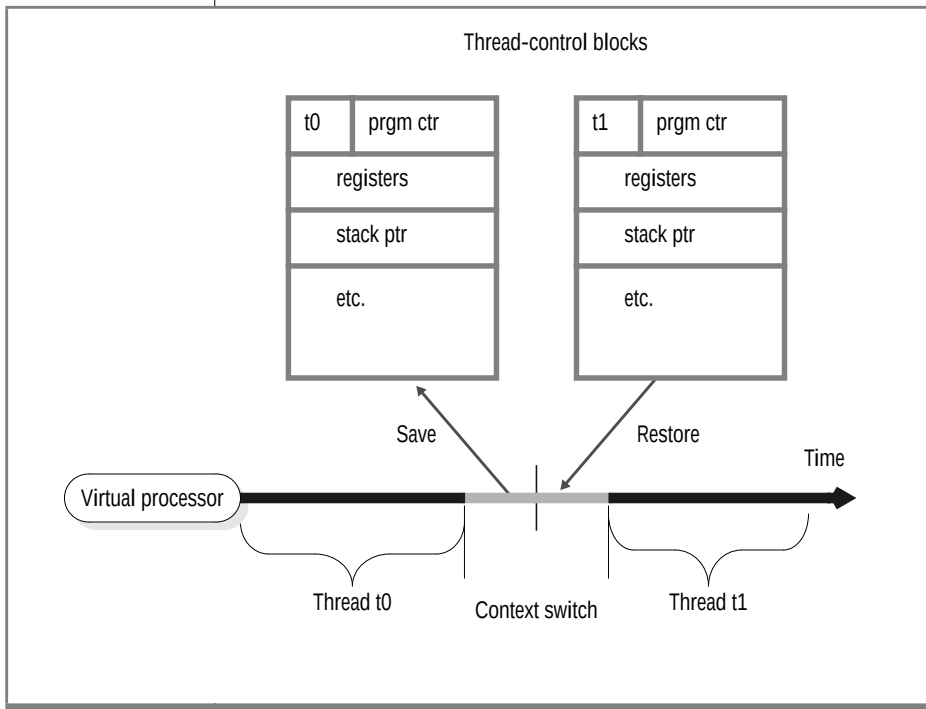


Figure 10-5
*Context Switch:
How a Virtual
Processor Switches
from One Thread to
Another*

Stacks

The 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 12-31](#).

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 10-6 illustrates how a virtual processor uses the stack to segregate nonshared data for session threads.

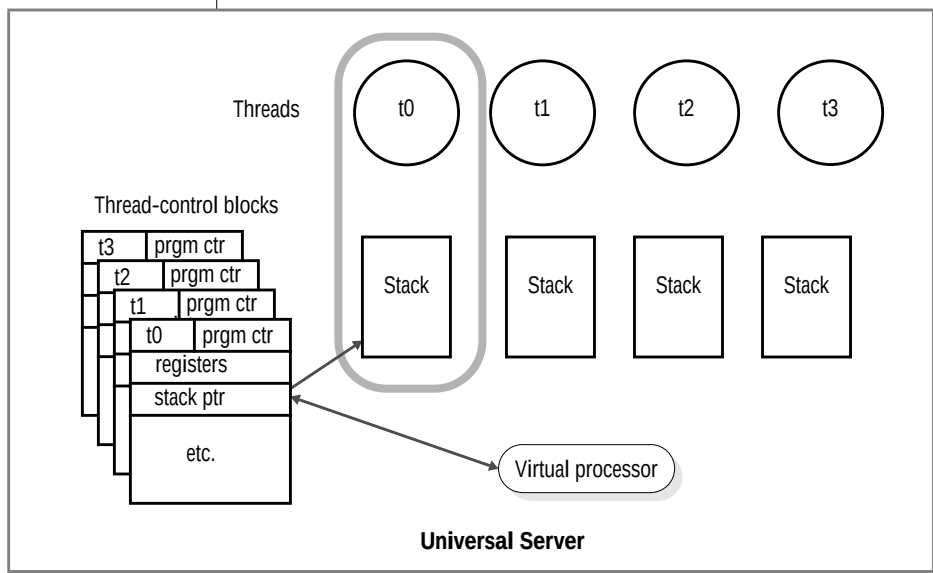


Figure 10-6
Virtual Processors
Segregate
Nonshared Data for
Each User

Queues

The 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 33-32](#). For information on how to add virtual processors while Universal Server is in on-line mode, refer to [“Adding Virtual Processors in On-Line Mode” on page 11-8](#).

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 10-7 illustrates how Universal Server threads are queued to perform database I/O.

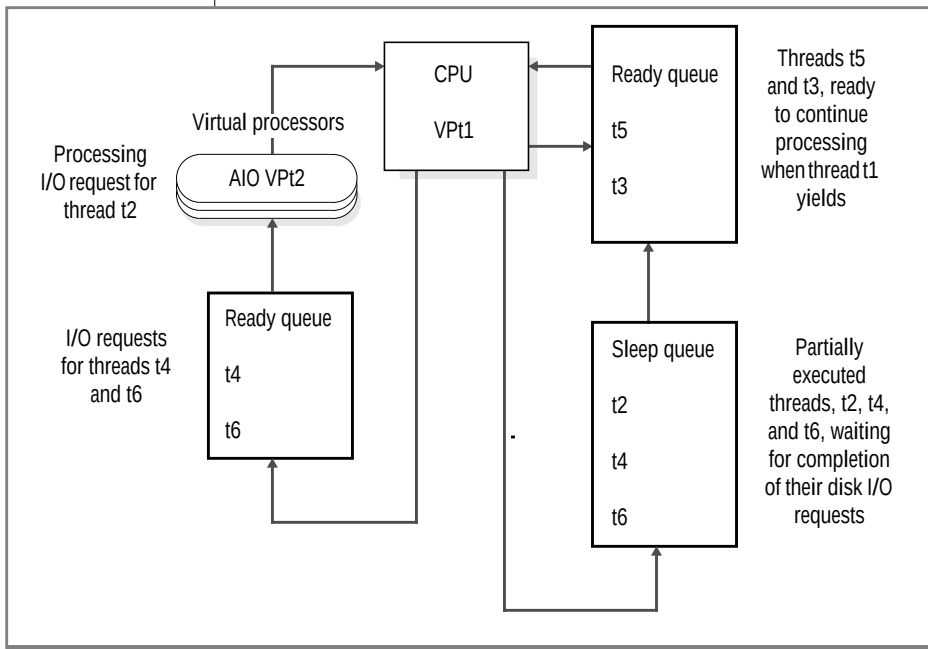


Figure 10-7
How Universal Server Threads Are Queued to Perform Database I/O

Wait Queues

Wait queues hold threads that need to wait for a particular event before they can continue to run. For example, wait queues coordinate access to shared data by threads. When a user thread tries to acquire the logical-log latch but finds that the latch is held by another user, the thread that was denied access puts itself in the logical-log wait queue. When the thread that owns the lock is ready to release the latch, it checks for waiting threads and, if threads are waiting, it wakes up the next thread in the wait queue.

Mutexes

A mutex (*mutually exclusive*) is a latching mechanism that Universal 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 33-27](#).

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 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 Universal 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 VPCLASS and NUMCPUVPS configuration parameters allow you to specify the number of CPU virtual processors that the database server starts initially. These parameters are mutually exclusive. If the ONCONFIG file contains a NUMCPUVPS parameter, you receive an error message if the file also contains a VPCLASS parameter. For information about these parameters, refer to [“VPCLASS” on page 37-75](#) and [“NUMCPUVPS” on page 37-51](#).

In addition to considering the number of CPUs in the computer and the number of users who will connect to the database server, you should also consider the fact that user-defined routines and DataBlade modules, which are collections of user-defined routines, run on either CPU virtual processors or user-defined virtual processors. For information on how you assign a user-defined routine to a virtual processor, refer to [“Assigning a User-Defined Routine to a User-Defined Virtual Processor Class” on page 10-25](#). For an additional consideration in deciding how many CPU virtual processors you need, refer to [“Should Poll Threads Run on CPU or Network Virtual Processors?” on page 10-32](#).

To evaluate the performance of the CPU virtual processors while Universal 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 37-46](#).

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 Universal 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 37-46](#).

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 37-68](#).

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.*



Setting `SINGLE_CPU_VP` parameter to 1 imposes two important restrictions on the database server, as follows:

- Only one CPU virtual processor is allowed.
- No user-defined classes are allowed. (However, users can still define routines that run directly on the CPU VP.)

Adding and Dropping CPU Virtual Processors in On-Line Mode

You can add or drop CPU class virtual processors while Universal Server is on-line. For instructions on how to do this, see [“Adding Virtual Processors in On-Line Mode” on page 11-8](#) and [“Dropping CPU or User-Defined Virtual Processors in On-Line Mode” on page 11-10](#).

Preventing Priority Aging

Some UNIX 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 13](#) of the Introduction.

If your operating system allows you to disable priority aging, you can disable it by specifying `noage` for the priority entry on VPCLASS configuration parameters or by setting the NOAGE parameter. These parameters are mutually exclusive. If the ONCONFIG file contains a VPCLASS parameter with the `noage` option set, you receive an error if the file also contains a NOAGE parameter. For more information on these parameters, refer to [“VPCLASS” on page 37-75](#) and [“NOAGE” on page 37-49](#).

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 Universal Server platform, refer to the Universal Server machine-notes file, which is described in [“Documentation Notes, Release Notes, Machine Notes”](#) on page 13 of the Introduction.

Figure 10-8 illustrates the concept of processor affinity.

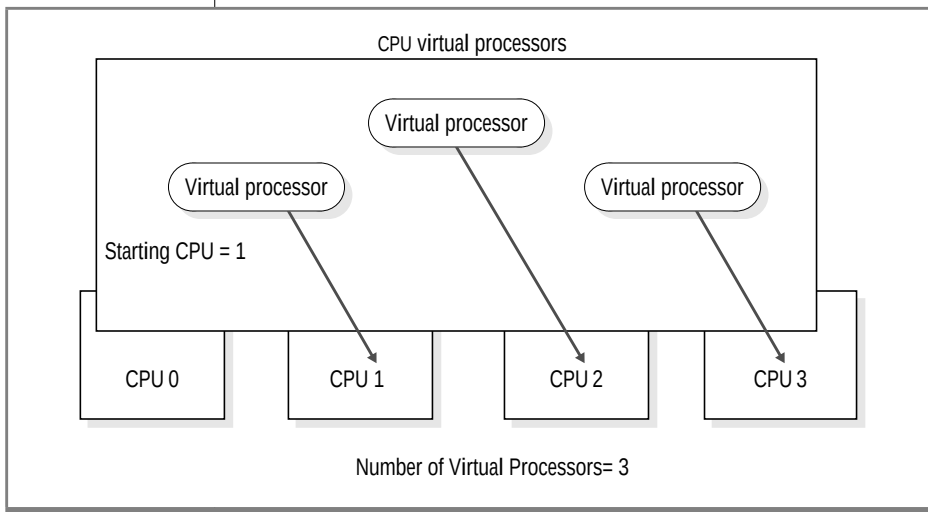


Figure 10-8
Processor Affinity

You can specify processor affinity by setting the affinity entry on the VPCLASS parameter for the CPU class of virtual processors or by setting the AFF_SPROC and AFF_NPROCS parameters. These parameters are mutually exclusive. If the ONCONFIG file contains a VPCLASS parameter that specifies processor affinity, you receive an error if the file also contains a AFF_SPROC and AFF_NPROCS parameters.

Setting Processor Affinity with the VPCLASS Parameter

To set processor affinity using the VPCLASS parameter, specify the range of CPUs to which you are assigning the virtual processors. In the following example, the affinity entry assigns the three virtual processors to CPUs 1, 2, and 3.

```
VPCLASS CPU,num=3,aff=1-3
```

For more information about the VPCLASS configuration parameter, refer to [“VPCLASS” on page 37-75](#).

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. Universal 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, Universal Server would display an error message because 3 (AFF_NPROCS) plus 2 (AFF_SPROC) equals 5, and there are only 4 CPUs available.

User-Defined Classes of Virtual Processors

You can designate classes of virtual processors, which are not predefined, to run user-defined routines and DataBlade modules. User-defined routines are typically written to support user-defined data types. If you do not want a user-defined routine to run in the CPU class, which is the default, you can assign it to a user-defined class of virtual processors (VPs). User-defined classes of virtual processors are also called *extension virtual processors*.

The Purpose of User-Defined Virtual Processors

The purpose of providing user-defined classes of virtual processors is to protect the database server from *ill-behaved* user-defined routines. If a CPU VP executes an ill-behaved routine, the database server is likely to crash or behave erratically. An ill-behaved user-defined routine has at least one of the following characteristics:

- Does not yield control to other threads
- Makes blocking operating-system calls
- Modifies the global VP state

You should assign any ill-behaved user-defined routines to a user-defined class of virtual processors. User-defined VPs relieve developers of the following restrictions on the CPU VP class:

- The need to yield the processor regularly
- The need to eliminate direct file-system access.

Functions that run in a user-defined virtual processor class need not yield the processor, and they might issue direct file-system calls that block further processing by the virtual processor until the I/O is complete. The normal processing of user queries is not affected because these are not CPU virtual processors. For a more detailed discussion of ill-behaved routines, refer to the Routine Description section of the [Informix Guide to SQL: Syntax](#).

Specifying a Nonyielding User-Defined Virtual Processor

You can cause a user-defined virtual processor class to run user-defined routines without yielding. You should assign a user-defined routine to a non-yielding class of virtual processors if the routine has not been designed or coded to handle the concurrency issues of multiprocessing. For example, if a user-defined routine uses global variables and makes calls to database services, such as the smart-large-object interface, you should assign it to a nonyielding class of user-defined virtual processors. Similarly, a call to `Thus`, another thread can run the same code and change the states of variables that the original thread assumes will be stable when it resumes processing.

To designate a user-defined class of virtual processors as nonyielding, specify the **noyield** option on the `VPCLASS` parameter. For more information on the **noyield** option, refer to [“VPCLASS” on page 37-75](#).

Assigning a User-Defined Routine to a User-Defined Virtual Processor Class

The SQL `CREATE FUNCTION` statement registers a user-defined routine. For example, the following `CREATE FUNCTION` statement registers the user-defined routine, **GreaterThanOrEqualTo**, and specifies that calls to this routine should be executed by the user-defined VP class named `UDR`:

```
CREATE FUNCTION GreaterThanOrEqualTo(ScottishName, ScottishName)
  RETURNS boolean
  WITH CLASS = UDR
```

In order to execute this function, the `ONCONFIG` file must include a line that configures the `UDR` class. If not, calls to the **GreaterThanOrEqualTo** function will fail. To configure the `UDR` class, include a line similar to the following one in the `ONCONFIG` file:

```
VPCLASS  UDR,num=2,noage
```

For more information on the `CREATE FUNCTION` statement, refer to the [Informix Guide to SQL: Syntax](#).

Adding and Dropping User-Defined Virtual Processors in On-Line Mode

You can add or drop virtual processors in a user-defined class while Universal Server is on-line. For instructions on how to do this, see [“Adding Virtual Processors in On-Line Mode” on page 11-8](#) and [“Dropping CPU or User-Defined Virtual Processors in On-Line Mode” on page 11-10](#).

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 10-28](#).

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, Universal Server prioritizes disk I/O by assigning different types of I/O to different classes of virtual processors and by assigning priorities to the non-logging 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 10-9 shows.

Figure 10-9
How Universal 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 Universal Server use a KAIO thread in the CPU virtual processor to perform I/O to the physical log. The physical-log file stores *before-images* of dbspace pages that have changed since the last *checkpoint*. (For more information on checkpoints, refer to [“Checkpoints” on page 12-56](#).) 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, Universal Server runs only one PIO virtual processor. If the physical-log file is in a dbspace that *is* mirrored, Universal Server runs two PIO virtual processors. This class of virtual processors has no parameters associated with it.

Asynchronous I/O

Universal 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

Universal 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 Universal 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 Universal Server machine-notes file, which is described under [“Documentation Notes, Release Notes, Machine Notes” on page 13](#) 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 either the VPCLASS parameter or the NUMAIOVPS parameter (but not both) to specify the number of AIO virtual processors that the database server starts initially. For information about these parameters, refer to [“VPCLASS” on page 37-75](#) and [“NUMAIOVPS” on page 37-50](#).

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 11-8](#).

You cannot drop AIO virtual processors while Universal 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 33-32](#).

If the database server implements kernel-asynchronous I/O on your platform, and all of your dbspaces are composed of raw file 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 Universal 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 Universal Server in two ways:

- Through shared memory
- Through a network connection

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*). For more information, refer to [“Using a Local Loopback Connection” on page 4-48.](#)

Universal Server provides four classes of network virtual processors to support the different types of client connections (for example, ipcshm, soctcp, tlitcp, and tlispx), as Figure 10-10 shows.

Figure 10-10
Universal Server Network Virtual Processors

Network VP	Connection Type
SHM	Shared memory (local only)
SOC	TCP/IP using sockets (local or remote)
STR	Stream pipe (local only)
TLI	TCP/IP using Transport Layer Interface (local or remote) or IPX/SPX using Transport Layer Interface (TLI)



Important: *Not all platforms support all connection types. For a list of network-interface protocols on your platform, refer to the machine-notes file.*

On a UNIX computer, Universal Server supports either the *sockets* or the *TLI* network interface, depending on which interface the platform supports. To determine which interface the database server supports on your platform, see the Universal Server machine-notes file, which is described under [“Documentation Notes, Release Notes, Machine Notes” on page 13](#) of the Introduction. The IPX/SPX protocol enables a Novell NetWare client to connect to Universal Server.

Specifying Network Connections

In general, the DBSERVERNAME and DBSERVERALIASES parameters define dbservernames that have corresponding entries in the **\$INFORMIXDIR/etc/sqlhosts** file. Each dbservername parameter in the **sqlhosts** file has a **nettype** entry that specifies an interface / protocol combination. The database server runs one or more *poll threads* for each unique **nettype** entry defined by a dbservername in the **sqlhosts** file. For a description of the **nettype** field, refer to [“The nettype Field” on page 4-27](#).

The NETTYPE configuration parameter provides optional configuration information for an interface / protocol combination. It allows you to allocate more than one poll thread for an interface / protocol combination and also designate the virtual processor class (CPU or NET) that will run the poll threads. For a complete description of this parameter, refer to [“NETTYPE” on page 37-47](#).

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 (SHM, SOC, TLI), on network virtual processors. In general, and particularly on a single-processor computer, poll threads run more efficiently on CPU virtual processors. This might not be true, however, on a multiprocessor computer with a large number of remote clients.

The NETTYPE parameter has an optional entry, called `vp class`, that allows you to specify either CPU or NET, for CPU or network virtual processor classes, respectively.

If you do not specify a `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 Universal 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 the **sqlhosts** file. For example, the **sqlhosts** file entry shown in Figure 10-11 causes the Universal Server **soc_ol1** to start a listen thread for **port1** on the host, or network address, **myhost**.

Figure 10-11
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 (SHM, SOC, or TLI). For information on the number of poll threads, refer to [“Specifying Network Connections” on page 10-32](#).

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 10-32](#) and [“NETTYPE” on page 37-47](#).

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 Universal Server, and starts an **sqlexec** thread, the session thread that does the primary processing for the client. Figure 10-12 illustrates the roles of the listen and poll threads in establishing a connection with a client application.

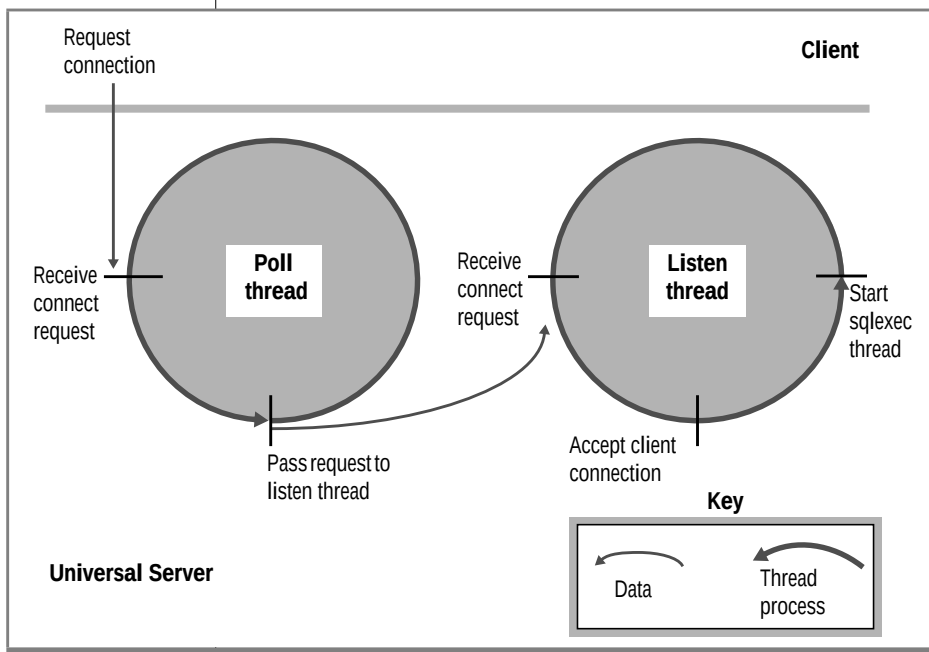


Figure 10-12
The Roles of the Poll
and the Listen
Threads in
Connecting to a
Client

A poll thread waits for requests from the client and places them in shared memory to be processed by the **sqlexec** thread. For a shared-memory connection, the poll thread places the message in the communications portion of shared memory. 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.

[Figure 10-13 on page 10-36](#) illustrates the basic tasks that the poll thread and the **sqlexec** thread perform in communicating with a client application.

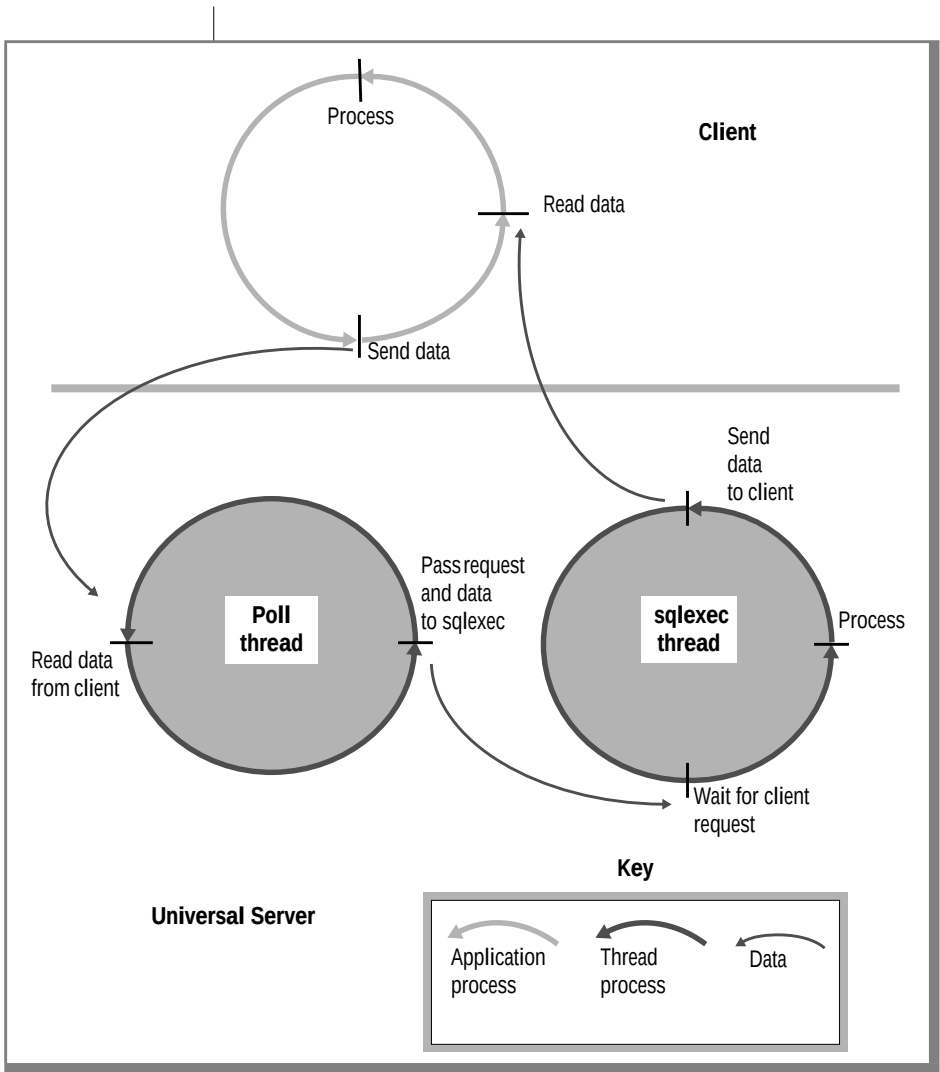


Figure 10-13
*The Roles of the Poll
and sqlexec Threads
in Communicating
with the Client
Application*

Starting Multiple Listen Threads

If Universal 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, Universal 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 10-14 defines two additional dbservernames, **soc_ol2** and **soc_ol3**, for the Universal Server instance identified as **soc_ol1**.

```
DBSERVERNAME      soc_ol1
DBSERVERALIASES   soc_ol2,soc_ol3
```

Figure 10-14
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. Each port is identified by a unique combination of **hostname** and **servicename** entries. For example, the **sqlhosts** file entries shown in [Figure 10-15 on page 10-38](#) cause Universal Server to start three listen threads for the **onsoctcp** interface / protocol combination, one for each of the ports defined.

Figure 10-15

Sqlhosts File 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 10-15, it applies to all of the connections shown. In this example, Universal 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, refer to [“The \\$INFORMIXDIR / etc / sqlhosts File” on page 4-24](#).

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 Universal 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 the **sqlhosts** file. For example, using the same dbservernames shown in Figure 10-14, the **sqlhosts** file entries shown in [Figure 10-16 on page 10-39](#) cause Universal Server to start three listen threads for the same interface / protocol combination (as did the entries in Figure 10-15). 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 10-16

Example of sqlhosts File 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	

Optical Virtual Processor

The optical class (OPT) of virtual processors is used only with INFORMIX-OnLine / Optical. INFORMIX-OnLine / Optical starts one virtual processor in the optical class if the STAGEBLOB configuration parameter is present. For more information on INFORMIX-OnLine / Optical, see the [INFORMIX-OnLine/Optical User Manual](#).

Audit Virtual Processor

Universal 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 Universal Server auditing, refer to the [INFORMIX-Universal Server 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.

Managing Virtual Processors

Setting Virtual-Processor Configuration Parameters	11-3
Setting Virtual-Processor Configuration Parameters with ON-Monitor	11-4
Setting Virtual-Processor Configuration Parameters with a Text Editor.	11-6
Starting and Stopping Virtual Processors	11-7
Adding Virtual Processors in On-Line Mode.	11-8
Using onmode to Add Virtual Processors in On-Line Mode . . .	11-8
Using ON-Monitor to Add Virtual Processors in On-Line Mode	11-9
Adding Network Virtual Processors	11-10
Dropping CPU or User-Defined Virtual Processors in On-Line Mode	11-10

T

his chapter describes how to set the configuration parameters that affect INFORMIX-Universal 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 10, “What Is the Universal Server Architecture?”](#)

Setting Virtual-Processor Configuration Parameters

You can set the configuration parameters for Universal Server virtual processors as **root** or user **informix**, using the following tools:

- ON-Monitor
- A text editor

ON-Monitor does not allow you to change all configuration parameters. For more information refer to [Chapter 37, “Configuration Parameters.”](#) This chapter provides descriptions of each configuration parameter, including the ON-Monitor menu sequences that lead you to the screen entries for each parameter.

Regardless of which method you use, you must reinitialize shared memory to put the changes into effect. For information on how to reinitialize shared memory, refer to [“Reinitializing Shared Memory” on page 13-14.](#)

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.

The VPCLASS configuration parameter consolidates information from the NUMCPUVPS, NUMAIOVPS, AFF_SPROC, and AFF_NPROCS parameters. Informix suggests that you use VPCLASS instead of those parameters.

Important: You cannot set VPCLASS using ON-Monitor; you can only set VPCLASS using a text editor. For more information on the VPCLASS parameter, refer to “VPCLASS” on page 37-75.



```
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]

Disable Priority Aging   [NOAGE]

NETTYPE settings:
Protocol Threads Users VP-class
[ipcshm]               [NETTYPE]
[socketp]              [NETTYPE]
```

Figure 11-1
Partial View of
ON-Monitor
PerFormance
Screen with the
ONCONFIG
Parameter for the
Virtual-Processor
Entries

Figure 11-2 shows the full perFormance screen; the shaded entries set configuration parameters for virtual processors.

Figure 11-2
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]
Num Procs to Affinity	[0]
Proc num to start with	[0]
CPU VPs	[5]
AIO VPs	[1]
Single CPU VP	[N]
Use OS Time	[N]
Disable Priority Aging	[N]
Off-Line Recovery Threads	[10]
On-Line Recovery Threads	[1]
Num of LRUS queues	[8]

LRU Max Dirty	[60]
LRU Min Dirty	[50]
Checkpoint Interval	[300]
Num of Read Ahead Pages	[50]
Read Ahead Threshold	[20]

NETTYPE settings:

Protocol	Threads	Users	VP-class
[ipcshm]	[2]	[5]	[CPU]
[soctcp]	[2]	[5]	[]

Are you running on a multiprocessor machine?

Figure 11-1 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 Universal Server virtual processors, refer to [Figure 11-3 on page 11-6](#).

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 11-3 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 37, “Configuration Parameters.”](#)

Figure 11-3
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 37-7
AFF_SPROC	Specifies the first CPU (of AFF_NPROCS) to which a CPU virtual processor will be assigned	page 37-8
MULTIPROCESSOR	Specifies that you are running on a multiprocessor computer	page 37-46
NETTYPE	Specifies parameters for network protocol threads (and virtual processors)	page 37-47
NOAGE	Specifies no priority aging of processes by the operating system	page 37-49
NUMAIOVPS	Specifies the number of AIO virtual processors	page 37-50

(1 of 2)

Parameter	Purpose	Reference
NUMCPUVPS	Specifies the number of CPU virtual processors	page 37-51
SINGLE_CPU_VP	Specifies that you are running a single CPU virtual processor	page 37-68
VPCLASS	Specifies a class of virtual processors, the number that Universal Server should start, and the maximum number allowed.	page 37-75

(2 of 2)

For more information on how these parameters affect virtual processors refer to [“Virtual-Processor Classes” on page 10-19](#).

Starting and Stopping Virtual Processors

When you start the **oninit** process to start Universal 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. For descriptions of the virtual-processor classes, refer to [“Virtual-Processor Classes” on page 10-19](#).

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” on page 11-8](#).

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 or User-Defined Virtual Processors in On-Line Mode” on page 11-10](#).

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 Universal Server Modes” on page 39-37](#).

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, and user-designated classes to run user-defined routines. 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

You can use only the **onmode** utility to start additional virtual processors for a user-defined class. The user-defined class must first be defined on a VPCLASS parameter. For more information about user-defined virtual processors, refer to [“Assigning a User-Defined Routine to a User-Defined Virtual Processor Class” on page 10-25](#). For the format of the **onmode** command, refer to [“onmode: Mode and Shared-Memory Changes” on page 39-35](#).

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.

Using ON-Monitor to Add Virtual Processors in On-Line Mode

To use ON-Monitor to add virtual processors while Universal Server is in on-line mode, select Modes from the main menu, and then select Add-Proc.

Figure 11-4 shows the ON-Monitor Add-Proc screen, which allows you to add virtual processors in the following classes: CPU, AIO, LIO, PIO, and network.

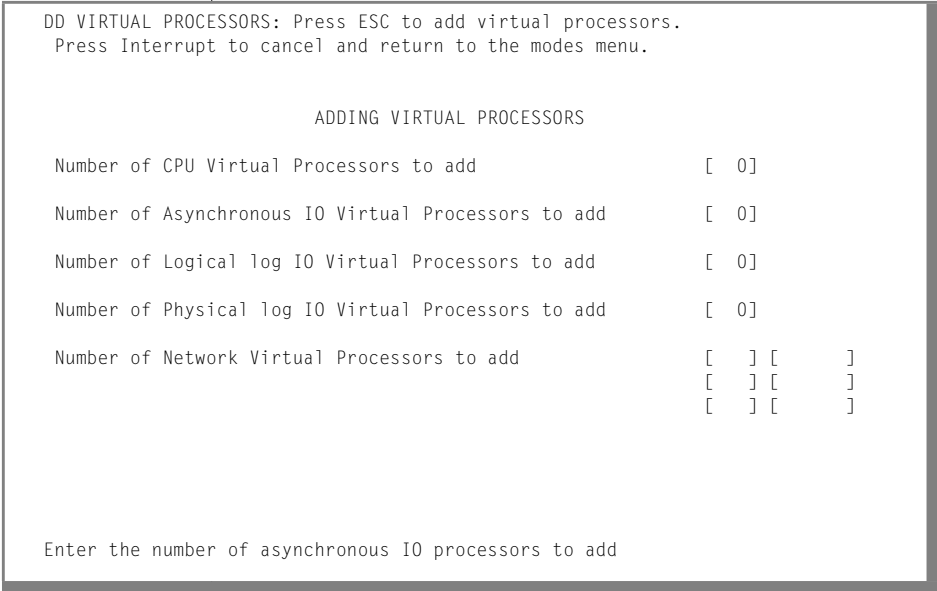


Figure 11-4
*The ON-Monitor
Add-Proc Screen*

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 11-8](#).

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 or User-Defined Virtual Processors in On-Line Mode

While Universal Server is in on-line mode, you can use the **-p** option of the **onmode** utility to drop, or terminate, CPU or user-defined virtual processors. Following the **onmode** command, specify a negative number that is the number of virtual processors that you want to drop, and then specify the 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 10-32.

Shared Memory

What Is Shared Memory?	12-5
How the Database Server Uses Shared Memory	12-6
How the Database Server Allocates Shared Memory	12-7
How Much Shared Memory Does the Database Server Need?.	12-9
What Action Should You Take If SHMTOTAL Is Exceeded?	12-10
What Processes Attach to Database Server Shared Memory?	12-11
How a Client Attaches to the Communications Portion	12-11
Where the Client Attaches to the Communications Portion	12-11
How Utilities Attach to Shared Memory	12-12
How Virtual Processors Attach to Shared Memory	12-12
Defining a Unique Key Value	12-13
Specifying Where to Attach the First Shared-Memory Segment	12-14
How Virtual Processors Attach Additional Shared-Memory Segments	12-15
The Shared-Memory Lower-Boundary Address	12-16
The Resident Portion of Database Server Shared Memory	12-17
Shared-Memory Header.	12-19
Shared-Memory Internal Tables	12-19
Hash Tables.	12-19
Buffer Table.	12-20
Chunk Table	12-20
Dbospace Table	12-21
Lock Table	12-21
Page-Cleaner Table	12-22
Tblspace Table.	12-23
Transaction Table.	12-23
User Table	12-24

Shared-Memory Buffer Pool	12-24
Regular Buffers	12-24
Logical-Log Buffer	12-26
Physical-Log Buffer	12-27
Data-Replication Buffer.	12-28
The Virtual Portion of Database Server Shared Memory	12-28
How the Database Server Manages the Virtual Portion of Shared Memory	12-28
How to Specify the Size of the Virtual Portion of Shared Memory	12-29
The Virtual Portion of Shared Memory	12-29
Big Buffers	12-29
Session Data	12-30
Thread Data.	12-30
Dictionary Cache	12-31
Sorting Memory	12-32
Stored Procedures Cache	12-33
Global Pool	12-33
The Communications Portion of Shared Memory	12-33
Virtual Extension Portion of Shared Memory	12-34
Concurrency Control	12-34
Shared-Memory Mutexes	12-34
Shared-Memory Buffer Locks	12-35
Types of Buffer Locks	12-35
How Database Server Threads Access Shared Buffers	12-36
LRU Queues	12-36
LRU Queue Components	12-36
Why Are Pages Ordered in Least-Recently Used Order?	12-37
LRU Queues and Buffer-Pool Management	12-37
How Many LRU Queues Should You Configure?	12-38
How Many CLEANERS Should You Allocate?	12-39
Limiting the Number of Pages Added to the MLRU Queues . .	12-39
When MLRU Cleaning Ends	12-40
Configuring the Database Server to Read Ahead	12-41
How a Database Server Thread Accesses a Buffer Page	12-42
Identify the Page	12-42
Determine the Level of Lock Access	12-42
Try to Locate the Page in Shared Memory	12-43
Locate a Buffer and Read Page from Disk	12-43

Lock the Buffer If Necessary	.12-43
Release the Buffer Lock and Wake a Waiting Thread	.12-44
How the Database Server Flushes Data to Disk	.12-45
Events That Prompt Flushing of the Regular Buffers	.12-46
Flushing Before-Images First	.12-46
Flushing the Physical-Log Buffer	.12-46
Events That Prompt Flushing of the Physical-Log Buffer	.12-47
When the Physical-Log Buffer Becomes Full	.12-48
How the Database Server Synchronizes Buffer Flushing	.12-49
Ensuring That Physical-Log Buffers Are Flushed First.	.12-49
How Write Types Describe Flushing Activity	.12-50
Foreground Write	.12-50
LRU Write	.12-51
Chunk Write	.12-51
Flushing the Logical-Log Buffer.	.12-52
When the Logical-Log Buffer Becomes Full	.12-52
After a Transaction Is Prepared or Terminated in a Database with Unbuffered Logging.	.12-53
When a Session That Uses Nonlogging Databases or Unbuffered Logging Terminates	.12-54
When a Checkpoint Occurs	.12-54
When a Page Is Modified That Does Not Require a Before-Image in the Physical-Log File.	.12-54
How the Database Server Achieves Data Consistency	.12-55
Critical Sections	.12-55
Checkpoints	.12-56
Events That Initiate a Checkpoint.	.12-56
Checkpoint Is Critical to Fast Recovery	.12-58
Time Stamps	.12-59
Time Stamps on Disk Pages	.12-59
Time Stamps on Blobpages	.12-59
Simple-Large-Object Time Stamps with Dirty Read and Committed Read Isolation Levels	.12-60
Buffering Large-Object Data	.12-61
Writing Simple-Large-Object Data	.12-61
Blobpages Do Not Pass Through Shared Memory	.12-61
Simple Large Objects Are Created Before the Data Row Is Inserted	.12-62
Blobpage Buffers Are Created for the Duration of the Write.	.12-62
Accessing Smart Large Objects	.12-64

T

his chapter describes the content of INFORMIX-Universal 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 Universal Server configuration parameters that determine shared-memory allocations, refer to [Chapter 13, “Managing Shared Memory.”](#)

What Is Shared Memory?

Shared memory is an operating-system feature that allows Universal Server threads and processes to share data by sharing access to pools of memory. Universal 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 processes read and write messages at the speed of memory transfers.

How the Database Server Uses Shared Memory

Universal 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 12-1 illustrates the shared-memory scheme.

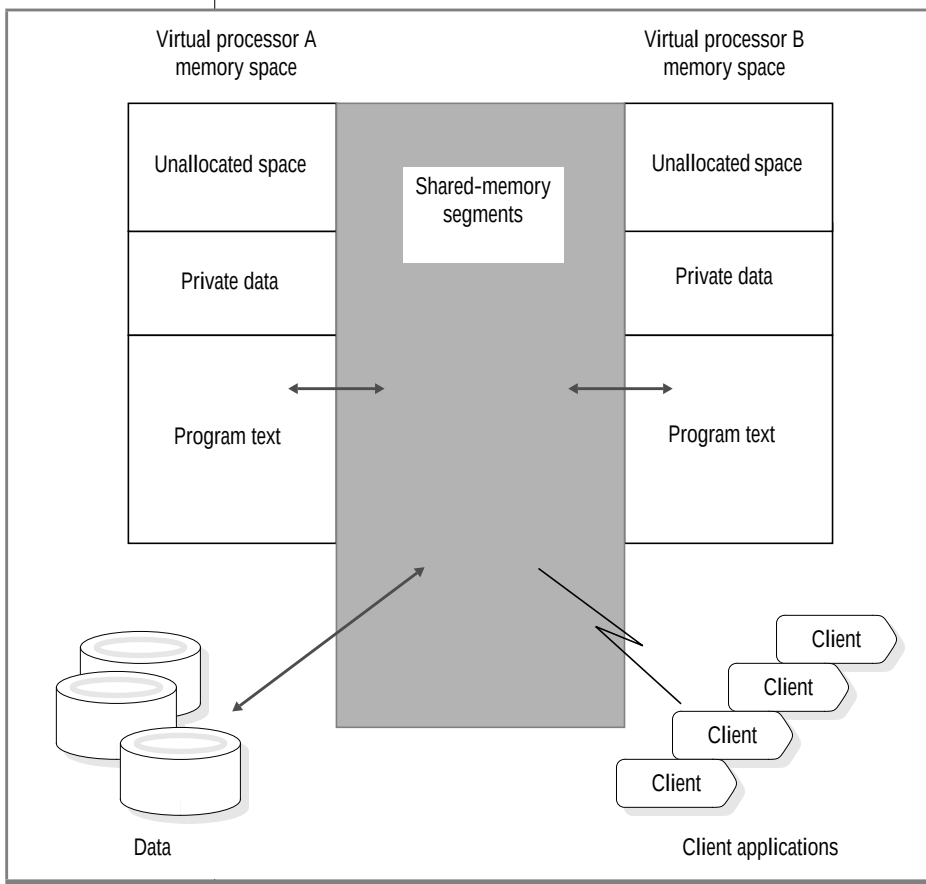


Figure 12-1
How the Database Server Uses Shared Memory

How the Database Server Allocates Shared Memory

Universal Server creates the following portions of shared memory:

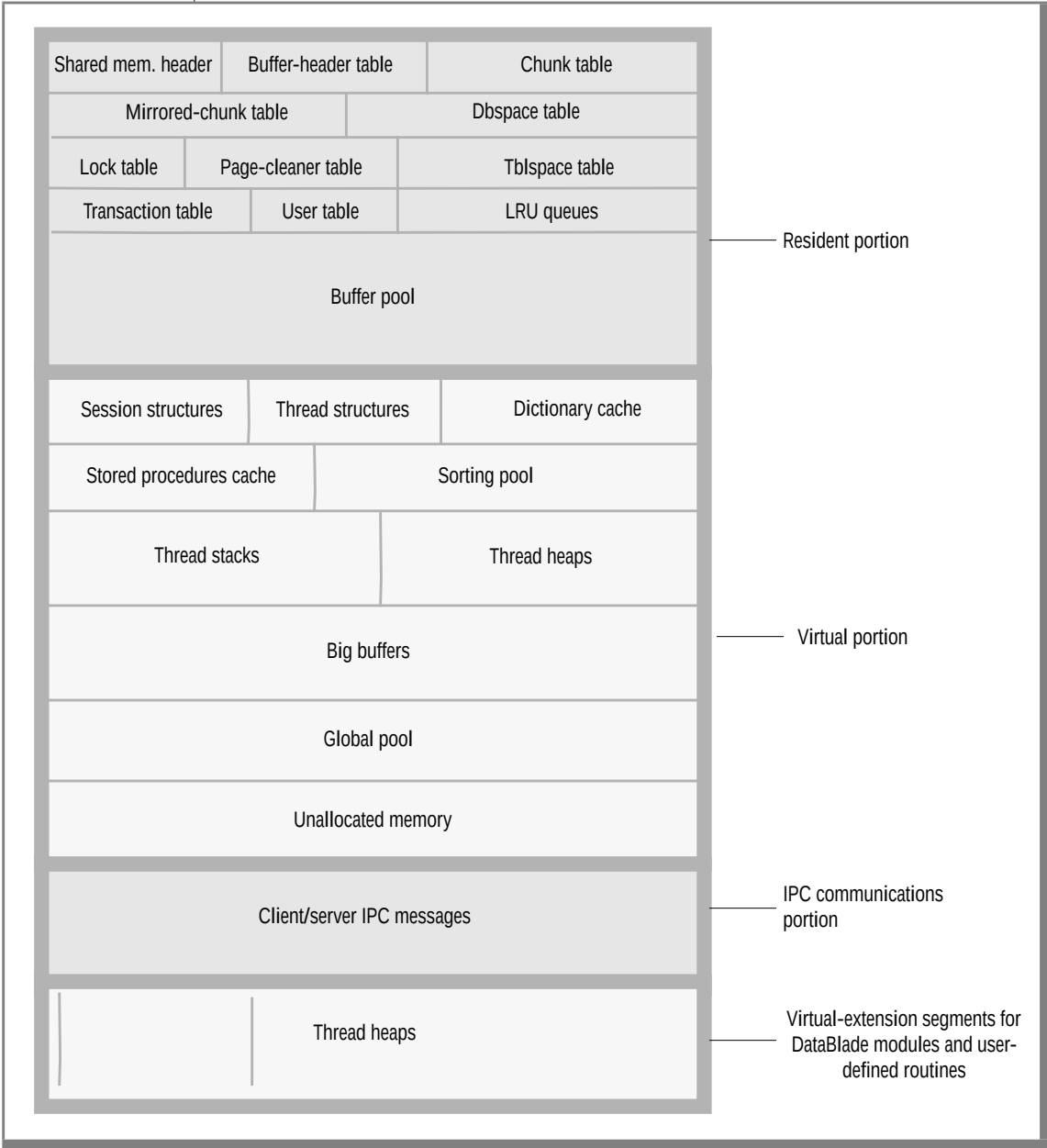
- The *resident* portion
- The *virtual* portion
- The *IPC communications* or message portion
- The *virtual extension* portion

Each portion of shared memory consumes one or more operating system segments. When Universal 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.

If the **sqlhosts** file specifies shared-memory communications, Universal Server allocates memory for the communications portion. Universal Server adds operating-system segments as needed to the virtual and virtual extension portions of shared memory. [Figure 12-2 on page 12-8](#) shows the contents of each portion of shared memory.

All Universal 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 Universal Server from off-line mode to quiescent mode, or from off-line mode directly to on-line mode. Universal 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, “What Are the Operating Modes?”](#)

Figure 12-2
Contents of Universal Server Shared Memory



How Much Shared Memory Does the Database Server Need?

Each portion of Universal 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 Universal Server shared memory, expressed in kilobytes. For information on how to use the **onstat** utility, refer to [“onstat: Monitor Universal Server Operation” on page 39-75](#). 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 Universal 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 is trying to perform an operation such as an index build or a hash join, it will return 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.

What Action Should You Take If SHMTOTAL Is Exceeded?

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: Universal Server may be unable to complete a checkpoint
17:19:13  Action: Make more virtual memory available to Universal Server
17:19:13  See Also: /tmp/af.c4
```

After informing you about the failure to allocate additional memory, the database server 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 Universal 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 Database Server Shared Memory?

The following processes attach to Universal Server shared memory:

- Client-application processes that communicate with Universal Server through the shared-memory communications portion (ipcshm)
- Universal Server virtual processors
- Universal Server utilities

The following sections describe how each type of process attaches to Universal Server shared memory.

How a Client Attaches to the Communications Portion

Client-application processes that communicate with Universal 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, [Chapter 4, “Client/Server Communications”](#) and [“Network Virtual Processors”](#) on page 10-31.

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 Universal Server shared memory), the user can set the **INFORMIXSHMBASE** environment variable to specify the address at which to attach Universal 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 Chapter 4 of the [Informix Guide to SQL: Reference](#).

How Utilities Attach to Shared Memory

Universal Server utilities such as **onstat**, **onmode**, and **ontape** attach to shared memory through the file **\$INFORMIXDIR/etc/infos.servername** where **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 **\$INFORMIXDIR/etc/infos.servername** when it starts the database server. The file is removed when the database server terminates.

How Virtual Processors Attach to Shared Memory

Universal 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

Universal 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
- UNIX returns the shared-memory identifier for the first shared-memory segment.

4. Directs UNIX 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:

$$(\text{SERVERNUM} * 65536) + \text{shmkey}$$

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 the UNIX operating system to attach the first shared-memory segment, it supplies the unique key value to identify the segment. In return, the UNIX 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 action 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 machine 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 UNIX systems accommodate additional memory at different virtual addresses. Some UNIX architectures extend the highest virtual address of the virtual-processor data segment to accommodate the next segment. In this case, the data segment might grow into the shared-memory segment.*
- *Some versions of UNIX require the user to specify a SHMBASE parameter of virtual address zero. The zero address informs the UNIX kernel that the kernel should pick the best address at which to attach the shared-memory segments. However, not all UNIX architectures support this option. Moreover, on some systems, the selection that the kernel makes might not be the best selection.*

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 is remaining. 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 UNIX 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:

$$\text{SHMBASE} + (\text{seg\_size} \times \text{number of attached segments})$$

The virtual processor repeats this process until it has acquired the total amount of shared memory.

Given the initial key value of $(\text{SERVERNUM} * 65536) + \text{shmkey}$, Universal Server can request up to 65,536 shared-memory segments before it could request a shared-memory key value used by another Universal 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 12-3 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. Since 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 UNIX kernel parameter that specifies the lower-boundary address or reconfigure the kernel to allow larger shared-memory segments. For a description of the UNIX kernel parameter, refer to [“The Role of the Shared-Memory Lower-Boundary Address”](#) on page 13-5.

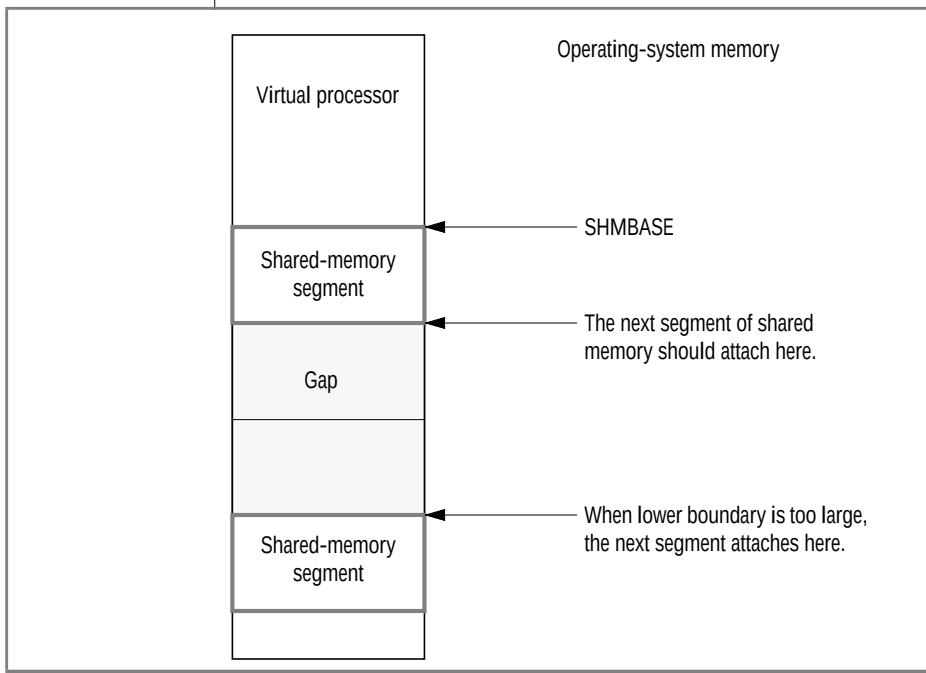


Figure 12-3
*Shared-Memory
Lower-Boundary
Address Overview*

The Resident Portion of Database Server Shared Memory

The UNIX 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/Os that would otherwise be required to access that data.

The resident portion of Universal Server shared memory stores the following data structures that do not change in size:

- Shared-memory header
- Internal tables
- Buffer pool

The database server requests that the operating system keep the resident portion 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.

Figure 12-4 illustrates the contents of the resident portion of shared memory.

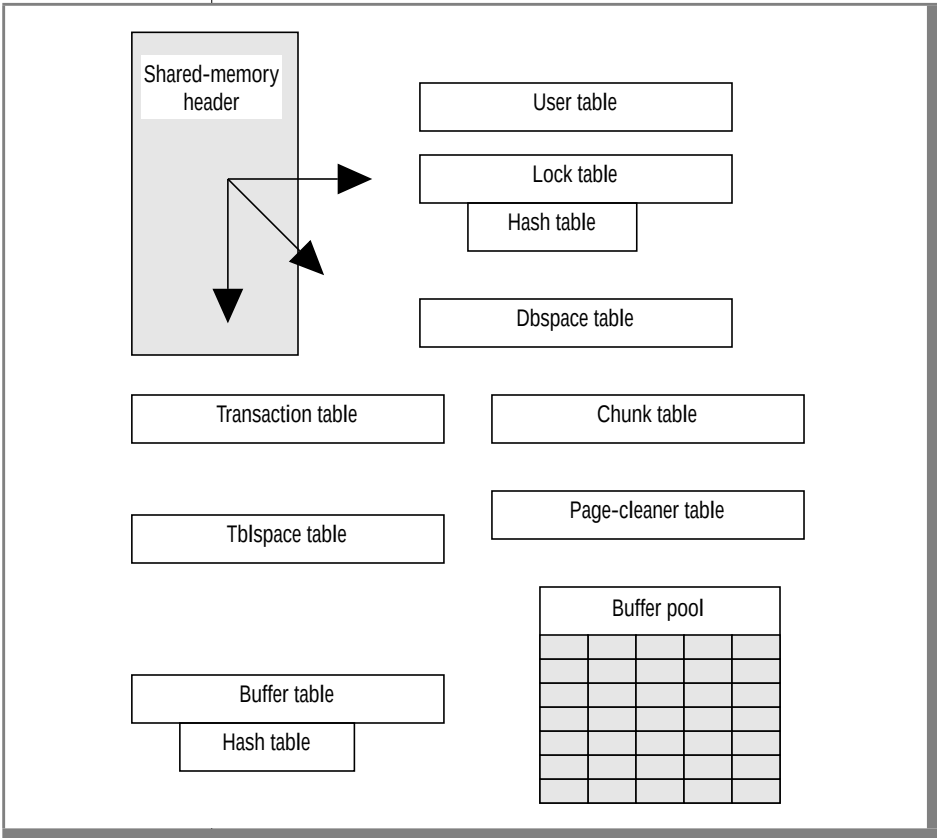


Figure 12-4
The Resident
Portion of Shared
Memory

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, although the size varies depending on the computer platform. You cannot tune the size of the header.

Shared-Memory Internal Tables

Universal 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
- Dbspaced table
- Lock table and associated hash table
- Page-cleaner table
- Tblspace table
- Transaction table
- User table

Hash Tables

Hashing is a technique that permits rapid lookup in tables where items are added unpredictably. The following 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 resident portion of shared memory.

Buffer Table

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 14-10](#).

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 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 33-21](#). For information on the database-server buffers, refer to [“Regular Buffers” on page 12-24](#).

Buffer-Table Hash Table

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 33-60](#).

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, blobspaces, and sbspaces 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 the dbspace was created

If the space is a blobspace, flags indicate the media where the blobspace is located—magnetic, removable media, or optical.

If the space is an sbspace, it contains internal tables that track smart-large-object metadata and large contiguous blocks of pages containing user data.

For information on monitoring dbspaces, refer to [“Monitoring the Database Server for Disabling I/O Errors” on page 33-58](#).

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 / or 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 37-32](#).

For information on monitoring locks, refer to [“Monitoring Locks” on page 33-29](#).

When Is a Byte Lock Generated?

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.

Hash-Table Entries

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 37-12](#).

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 Universal Server Operation” on page 39-75](#).

Tblspace Table

The tblspace table tracks all active tablespaces in a Universal 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 tablespaces 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 tablespaces on disk. (Do not confuse the shared-memory active tblspace table with the tblspace tablespaces.) For information on monitoring tablespaces, refer to [“Monitoring Tablespaces and Extents” on page 33-64](#).

Universal Server manages one tblspace table for each dbspace.

Transaction Table

The transaction table tracks all transactions in the database server. 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.

Tracking information derived from the transaction table appears in the **onstat -x** display. For an example of the output displayed by **onstat -x**, [“Monitoring Transactions” on page 33-45](#).

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](#).

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 system threads include one to monitor and control checkpoints, one to process **onmode** commands, the b-tree cleaner and one to many page cleaners.

The database server increases the number of entries in the user table as needed. You can monitor user threads using the **onstat -u** command.

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 using 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 maximum number of allocated buffers is 768 kilobytes (768 * 1024). For more information on setting the **BUFFERS** configuration parameter, refer to [“BUFFERS” on page 37-9](#).

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 Universal Server I/O threads that are waiting for buffers.

For a description of how LRU queues work, refer to [“LRU Queues” on page 12-36](#). For a description of mutexes, refer to [“Mutexes” on page 10-18](#).

For information on specifying the number of buffers, refer to [“BUFFERS” on page 37-9](#).

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 12-29](#)) and I/O performed from blobpage buffers (see [“Blobpage Buffers Are Created for the Duration of the Write” on page 12-62](#)). To determine the page size for your system, select the shared-memory option of the Parameters menu in ON-Monitor. ON-Monitor displays a list of shared-memory parameters, of which the Universal Server page size is the last entry on the page. The **-b** option of the **onstat** utility also displays the Universal Server page size. For information on the **onstat** utility, refer to [“onstat: Monitor Universal Server Operation” on page 39-75](#).

Logical-Log Buffer

The database server uses the logical log to store a record of changes to Universal 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 12-5.

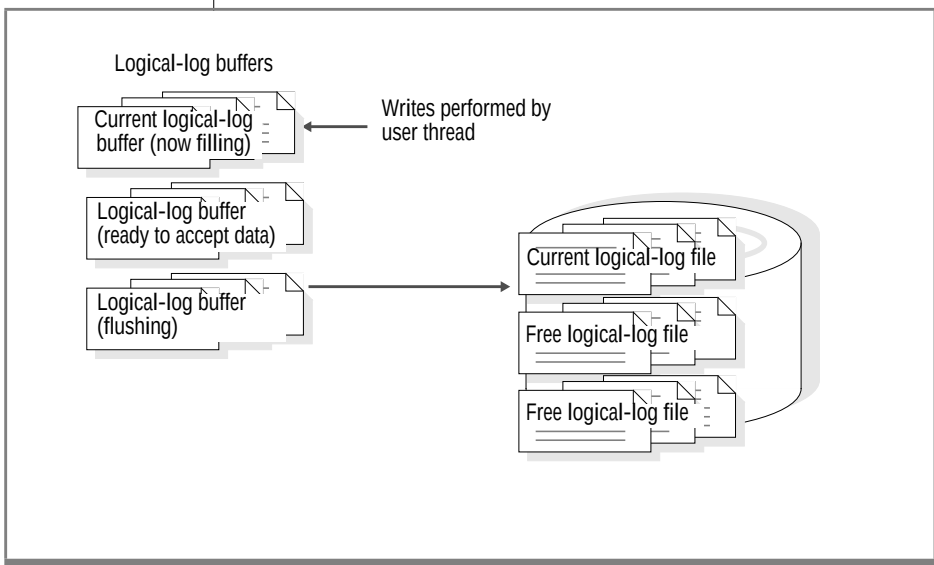


Figure 12-5
The Logical-Log Buffer and its Relation to the Logical-Log Files on Disk

For a description of how the database server flushes the logical-log buffer, refer to [“Flushing the Logical-Log Buffer” on page 12-52](#).

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, simple large objects in dbspaces). For the possible values that you can assign to this parameter, refer to [“LOGBUFF” on page 37-33](#).

For information on the impact of large objects on shared memory buffers, refer to [“Buffering Large-Object Data” on page 12-61](#)

Physical-Log Buffer

Universal 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 12-46](#).

For information on monitoring the physical-log file, refer to [“Monitoring the Physical-Log File” on page 33-51](#).

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 Universal 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 37-56](#).

Data-Replication Buffer

Data replication requires two instances of Universal Server, a primary instance of Universal Server and a secondary instance of Universal Server, running on two computers. If you implement data replication for your database server, the database server holds logical-log records in the data-replication buffers before it sends them to the secondary database server. A data-replication buffer is always the same size as the logical-log buffer. For information on the size of the logical-log buffer, refer to the preceding section, [“Logical-Log Buffer” on page 12-26](#). For more information on how the data-replication buffer is used, refer to [“How Does Data Replication Work?” on page 29-8](#).

The Virtual Portion of Database Server Shared Memory

The virtual portion of shared memory is expandable by the database 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.

In addition to the virtual portion of shared memory, Universal Server also uses a virtual extension portion for use by user-defined routines. For more information on the virtual extension portion, refer to [“Virtual Extension Portion of Shared Memory” on page 12-34](#).

How the Database Server Manages the Virtual Portion of Shared Memory

Universal 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 Universal 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 37-67](#), [“SHMADD” on page 37-64](#), and [“Adding a Segment to the Virtual Portion of Shared Memory” on page 13-15](#).

The Virtual Portion of Shared Memory

The virtual portion of shared memory stores the following data:

- Big buffers
- Session data
- Thread data (stacks and heaps)
- Dictionary cache
- Stored procedures cache
- Sorting pool
- Global pool

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 a simple large object 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 Universal Server Operation” on page 39-75](#).

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 these. 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 10-13](#).

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 10-15](#). For information on how to monitor the size of the stack for a session, refer to [“Monitoring Sessions and Threads” on page 33-35](#).

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 37-69](#).

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 only affects the stack space for 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 could occur concurrently, and the average row size is 200 bytes, and the average number of rows in a table is 400, you could estimate the amount of shared memory that the database server needs for sorting as follows:

$$30 \text{ sorts} * 200 \text{ bytes} * 400 \text{ rows} = 2,400,000 \text{ bytes}$$

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 using the following formula:

$$\text{quantum} = \text{DS\_TOTAL\_MEMORY} / \text{DS\_MAX\_QUERIES}$$

When MGM controls the resources for a query, sorting will 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 will wait until memory is available. For more information about MGM, see [Chapter 18, “What Is PDQ?”](#)

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 the [INFORMIX-Universal Server Performance Guide](#).

If PDQ priority is 0, the maximum amount of shared memory that Universal 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 (**nettype**: `socket`, `tlitcp`, or `ipx/spx`) deposit messages from clients. The **sqlxexec** threads pick up the messages from here and process them.

The Communications Portion of Shared Memory

Universal 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 12-11](#).

Virtual Extension Portion of Shared Memory

The virtual extension portion of shared memory contains additional virtual segments and virtual extension segments. Virtual extension segments contain thread heaps for DataBlade modules and user-defined routines that run in user-defined virtual processors.

The SHMADD and SHMTOTAL configuration parameters apply to the virtual extension portion of shared memory, just as they do to the other portions of shared memory.

Concurrency Control

Universal 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 holding 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 33-27](#).

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 33-29](#).

Detailed information about locking and process isolation during SQL processing is provided in the [Informix Guide to SQL: Tutorial](#). For further information about locking and shared memory, refer to “Shared-Memory Buffer Locks” on this page.

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 12-21](#) and [“Shared-Memory Buffer Locks” on page 12-35](#).

How Database Server Threads Access Shared Buffers

Universal 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 Universal 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

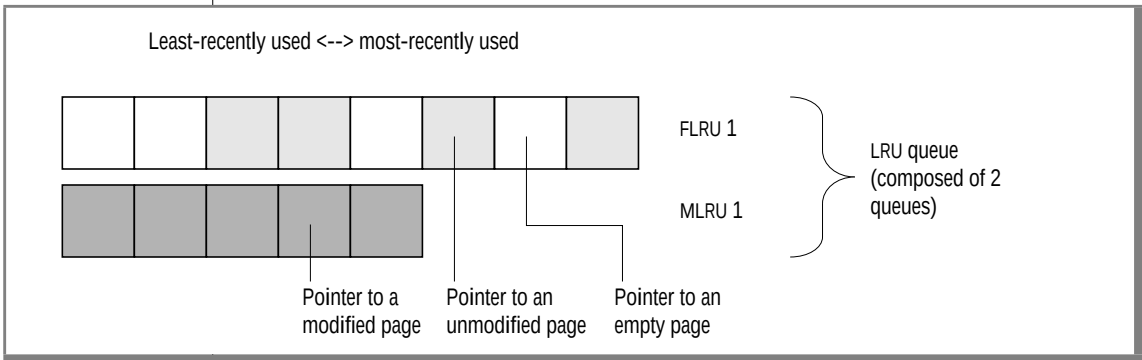
The LRU queue is composed of two queues, the FLRU and the MLRU queues.

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 / 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 12-6 illustrates the structure of the LRU queues.

Figure 12-6
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, Universal 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 33-24](#).

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:

$$\text{LRUS} = \max(4, (\text{no. of cpu VPs}))$$

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, this is only true if you 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 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 12-45](#).

Figure 12-7 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 12-7
*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 12-8 on page 12-41](#) 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: $1000 \times .050 = 500$

Figure 12-8
*How
LRU_MIN_DIRTY
Specifies the Point
at Which Page
Cleaning Can End*

For more information on how Universal Server flushes the buffer pool, refer to [“How the Database Server Flushes Data to Disk” on page 12-45](#).

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 Universal 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 Universal Server use of read-ahead, refer to [“Monitoring Shared-Memory Profile” on page 33-20](#) and [“-p Option” on page 39-96](#) under the heading [“onstat: Monitor Universal Server Operation.”](#)

How a Database Server Thread Accesses a Buffer Page

Universal 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 12-35.](#))

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 remain 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 Universal 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 37-12](#).

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.
- Universal 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, 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 12-9 on page 12-47](#).

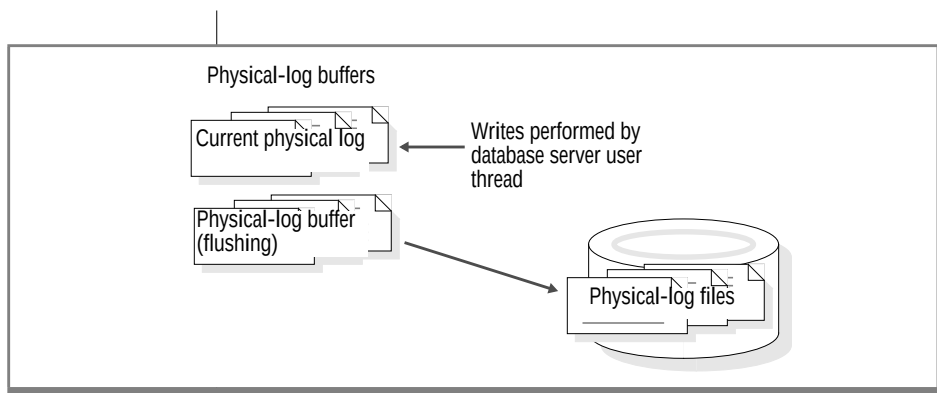


Figure 12-9
The Physical-Log Buffer and its Relation to the Physical Log on Disk

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 24, “What Is Physical Logging?”](#) For a description of fast recovery, refer to [Chapter 26, “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 12-55](#).

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 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. If this is the 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 10-16.](#)) 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 12-36.](#)

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 39-87.](#)

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 33-24.](#)

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 12-36.](#))

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 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 are performing a query are freed from having to flush a page to disk before reading 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 done 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 22, “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 37-42](#).

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 12-56](#).

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

Universal 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 26, “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.
- The database server administrator initiates a checkpoint from the ON-Monitor, Force-Ckpt menu or from the command line using **onmode -c**.
- Certain administrative tasks, such as adding a chunk or a dbspace, take place.

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 checkpoint record to 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 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.

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 42-34](#).

Time Stamps on Blobpages

In addition to the page-header and page-ending time-stamp pair, each disk page that contains a simple large object also contains one member of a second pair of time stamps. This second pair of time stamps is the simple large object time-stamp pair. The time stamp that appears on the disk page where the simple large object is stored is paired with a time stamp that is stored with the forward pointer to this simple large object segment. The forward pointer is stored either in the data row (with the descriptor) or with the previous segment of simple large object data. For more information on time stamps on blobpages, refer to [“Blobspace Structure and Storage” on page 42-62](#).

A simple-large-object time-stamp pair is updated whenever a simple-large-object column is updated. When a simple large object in a data row is updated, the new simple large object is stored on disk, and the forward pointer stored with the descriptor is revised to point to the new location. The simple-large-object time stamp in the data row is updated and synchronized with the simple-large-object time stamp on the disk page of the new simple large object.

Simple-Large-Object Time Stamps with Dirty Read and Committed Read Isolation Levels

Because retrieving a simple large object can involve large amounts of data, it might be impossible to retrieve the simple-large-object data simultaneously with the rest of the row data. Coordination is needed for simple-large-object reads at the Dirty Read or Committed Read level of isolation. Therefore, each read compares the two members of the simple-large-object 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 simple-large-object forward pointer read by the user thread is no longer valid.

To understand how a forward pointer stored with a simple-large-object descriptor or with the previous segment of simple-large-object 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 one user thread is deleting a simple large object from a data 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 simple large object is deleted, the space is freed, and a third thread starts to write new simple-large-object data in the newly freed space where the first simple large object was stored. Eventually, when the second user thread starts to read the simple-large-object data at the location where the first simple large object was stored, the thread compares the time stamp from the simple-large-object descriptor with the time stamp that precedes the simple-large-object data. The time stamps do not match. The simple-large-object 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 the space on the disk page could be reused. If the space is reused in the interim, the simple-large-object time stamp 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 Large-Object Data

You can assign simple large objects (BYTE and TEXT data types) to either dbspaces or blobspaces. You can assign smart large objects (CLOB and BLOB data types) only to sbspaces. The database server uses different methods to access each type of space.

Writing Simple-Large-Object Data

The database server writes simple-large-object data that is stored in a dbspace to disk pages 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 12-45](#).

The database server writes simple-large-object data to a blobspace very differently than the way 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 42-62](#).

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 blobpage 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, Universal 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 [“Blobpage Logging” on page 22-28](#).

Blobpages stored on optical media are not written to dbspace and logical-log backup tapes due to the high reliability of optical media.

Simple Large Objects Are Created Before the Data Row Is Inserted

When simple-large-object data is written to disk, the row itself might not exist yet. During an insert, for example, the large object is transferred before the rest of the row data. After the simple large object is stored, the data row is created with a 56-byte descriptor that points to the location of the large object. For a description of how simple large objects are stored physically, refer to [“Simple-Large-Object Storage and the Simple-Large-Object Descriptor” on page 42-64](#).

Blobpage Buffers Are Created for the Duration of the Write

To receive simple-large-object data from the application process, the database server creates a pair of blobpage buffers, one for reading and one for writing, each the size of one blobpage. Each user has only one set of simple-large-object buffers and, therefore, can access only one simple large object at a time.

Simple-large-object data is transferred from the client-application process to the database server in 1-kilobyte segments. The database server begins filling the blobpage buffers with the 1-kilobyte pieces and attempts to buffer two blobpages at a time. The database server buffers two blobpages so 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 blobpage 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 UNIX kernel loops internally (in kernel mode) until the transfer is complete.

The blobpage buffers remain until the thread that created them is finished. When the simple large object is written to disk, the database server deallocates the pair of blobpage buffers. Figure 12-10 illustrates the process of writing a simple large object to a blobpage.

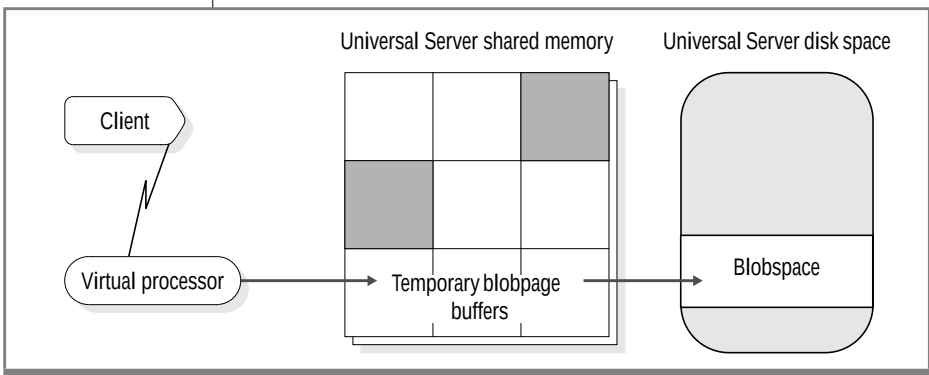


Figure 12-10
Writing a Simple Large Object to a Blobpage

Tracking Blobpages

Blobspace blobpages are allocated and tracked using the free-map page. Links connecting the blobpages and pointers to the next blobpage segments are created as needed.

A record of the operation (insert, update, or delete) is written to the logical-log buffer.

Accessing Smart Large Objects

Universal Server accesses smart large objects through the regular shared-memory buffers, in the same way that it accesses data that is stored in a dbspace. The user-data portion of a smart large object is buffered at a lower priority than normal buffer pages, however, to prevent flushing data of higher value out of the buffer pool. Buffering permits faster access to smart large objects that are accessed frequently.

A smart large object is stored in an sbpace. You cannot store simple large objects in sbspaces, and you cannot store smart large objects in a **blob**space. An sbpace consists of a user-data portion and a metadata portion. The user-data portion contains the smart-large-object data. The metadata portion contains information about the content of the sbpace. For more information on the structure and content of an sbpace, refer to [“Sbpace Structure” on page 42-71](#).

Because smart large objects pass through the shared-memory buffer pool and can be logged, you must consider them when you allocate buffers. You use the BUFFERS configuration parameter to allocate regular shared-memory I/O buffers. You use the LOGBUFF configuration parameter to specify the size of the logical-log buffer. You can find information about setting each of these parameters on the following pages:

- [BUFFERS](#) [page 37-9](#)
- [LOGBUFF](#) [page 37-33](#)

The user-data portion of smart large objects that are logged does not pass through the physical log, so the PHYSBUF parameter does not need to be adjusted for smart large objects.

For information on the structure of an sbpace, which contains smart large objects, refer to [Chapter 42, “Disk Structures and Storage.”](#) For information on creating an sbpace, refer to [“onspaces: Manage Blobspaces, Dbspaces, Extspaces, or Sbspaces” on page 39-55](#).

Managing Shared Memory

Setting Shared-Memory Configuration Parameters	13-3
UNIX Kernel Configuration Parameters	13-4
Shared-Memory Configuration Parameters	13-6
Setting Configuration Parameters for the Resident Portion of Shared Memory with ON-Monitor	13-7
Setting Configuration Parameters for the Resident Portion of Shared Memory with a Text Editor	13-8
Setting Configuration Parameters for the Virtual Portion of Shared Memory with ON-Monitor	13-10
Setting Configuration Parameters for the Virtual Portion of Shared Memory with a Text Editor	13-11
Setting Configuration Parameters for the Shared-Memory Performance Options with ON-Monitor	13-12
Setting Configuration Parameters for the Shared-Memory Performance Options with a Text Editor	13-13
Reinitializing Shared Memory	13-14
Turning Residency On or Off for Resident Shared Memory	13-14
Turning Residency On or Off in On-Line Mode.	13-14
Turning Residency On or Off for the Next Time You Reinitialize Shared Memory	13-15
Adding a Segment to the Virtual Portion of Shared Memory	13-15
Forcing a Checkpoint	13-16

T

his chapter tells you how to perform tasks related to managing the use of shared memory with INFORMIX-Universal Server. It assumes you are familiar with the terms and concepts contained in [Chapter 12, “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 Universal 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 [“Adjusting the Amount of Memory” on page 19-7.](#)

Setting Shared-Memory Configuration Parameters

You must consider the following two sets of configuration parameters when you configure Universal Server shared memory:

- UNIX kernel parameters
- Universal Server shared-memory configuration parameters

The following two sections describe the effects of these two sets of parameters in configuring Universal Server shared memory.

UNIX Kernel Configuration Parameters

Nine UNIX configuration parameters can affect the use of shared memory by Universal 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 shared-memory segment size, expressed in bytes or kilobytes
- Minimum shared-memory segment size, expressed in bytes
- Maximum number of shared-memory identifiers
- 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 UNIX environment, refer to the machine-notes file that is provided with Universal Server. For more information about the machine-notes file, refer to [“Documentation Notes, Release Notes, Machine Notes” on page -13](#).

Role of Maximum UNIX Shared-Memory Segment Size

When Universal Server creates the required shared-memory segments, it attempts to acquire as large an operating-system segment as possible. The first segment size Universal 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.

Universal 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 Universal 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 Universal Server creates as many additional segments as it requires.

Role of Maximum Shared-Memory Identifiers

Shared-memory identifiers affect Universal Server operation when a virtual processor attempts to attach to shared memory. UNIX identifies each shared-memory segment with a shared-memory identifier. For most UNIX 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 12-12.

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.

The Role of the Shared-Memory Lower-Boundary Address

When Universal 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 UNIX 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 12-12.

Guideline for Total Addressable Size per Process

Check that the maximum amount of memory that can be allocated is equal to the total addressable shared-memory size for a single operating-system process. The following equation expresses the concept another way:

$$\begin{aligned} \text{Maximum amount of shared memory} = \\ (\text{Maximum number of attached shared-memory segments per} \\ \text{process}) \times (\text{Maximum shared-memory segment size}) \end{aligned}$$

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.

Semaphore Guidelines

Universal Server operation requires 1 UNIX semaphore for each virtual processor, 1 for each user who connects to Universal Server through shared memory (ipcshm protocol), 6 for Universal Server utilities, and 16 for other purposes.

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:

- 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.

Setting Configuration Parameters for the Resident Portion of Shared Memory with ON-Monitor

To set the configuration parameters for the resident portion of shared memory using ON-Monitor, select Parameters from the main menu, and then select the Shared-Memory option. Figure 13-1 shows an example of a Shared-Memory screen. The shaded entries set configuration parameters for the resident portion of shared memory.

Figure 13-1
ON-Monitor 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   [ 0]      Server Name [odyssey_01]
Server Aliases [oddsoc_011,oddsoc_012,oddsoc_013]
Dbospace Temp   [
Deadlock Timeout [ 0] Secs
Forced Residency [Y]      Number of Page Cleaners [ 1]
Non Res. SegSize (Kbytes) [ 4000] Stack Size (Kbytes) [ 32]
Optical Cache Size (Kbytes)[ 0]

Physical Log Buffer Size [ 32] Kbytes
Logical Log Buffer Size [ 32] Kbytes
Max # of Logical Logs   [ 6]
Max # of Locks          [ 2000]
Max # of Buffers        [ 200]

Dbospace Down Option    [ 0]
Preserve Log for Log Backup [N]
Transaction Timeout     [ 300]
Long TX HWM             [ 50]
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-OnLine.
  
```



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 13-2 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]

Forced Residency    [Y]
Number of Page Cleaners[CLEANERS]

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-OnLine.
```

Figure 13-2
*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 13-3 on page 13-9](#).

Setting Configuration Parameters for the Resident Portion of 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 13-3 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 37, “Configuration Parameters.”](#)

Figure 13-3
Configuring the Resident Portion of Shared Memory

ONCONFIG Parameter	Purpose	Reference
BUFFERS	Specifies the maximum number of shared-memory buffers	page 37-9
CLEANERS	Specifies the number of page-cleaner threads that the database server is to run	page 37-12
LOCKS	Specifies the maximum number of locks for database objects; for example, rows, key values, pages, and tables	page 37-31
LOGBUFF	Specifies the size of the logical-log buffers	page 37-33
PHYSBUFF	Specifies the size of the physical-log buffers	page 37-56
RESIDENT	Specifies residency for the resident portion of the database server shared memory	page 37-60
SERVENUM	Specifies a unique identification number for the database server on the local host computer	page 37-63
SHMADD	Specifies the size of dynamically added shared-memory segments	page 37-64
SHMTOTAL	Specifies the total amount of memory to be used by the database server	page 37-65

Setting Configuration Parameters for the Virtual Portion of Shared Memory with ON-Monitor

To set the configuration parameters for the virtual portion of shared memory using ON-Monitor, select Parameters from the main menu and then select the Shared-Memory option. Figure 13-4 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_o]      ]
Server Aliases [cole,davison,stackhouse]
Dbospace Temp [
Deadlock Timeout      [ 60] Secs Number of Page Cleaners [ 1]
Forced Residency      [N]      Stack Size (Kbytes) [ 32]
Non Res. SeqSize (Kbytes) [ 4000] Optical Cache Size (Kbytes) [ 0]

Physical Log Buffer Size [ 32] Kbytes      Dbospace 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-OnLine.
```

Figure 13-4
*Setting Virtual
Shared-Memory
Configuration
Parameters*

Figure 13-5 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

Non Res. SeqSize (Kbytes) [SHMVIRTSIZE]      Stack Size (Kbytes) [STACKSIZE]
```

Figure 13-5
*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 13-6.

Setting Configuration Parameters for the Virtual Portion of 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 13-6 lists the ONCONFIG parameters that you use to configure the virtual portion of shared memory.

Figure 13-6
Configuring the Virtual Portion of Shared Memory

ONCONFIG Parameter	Purpose	Reference
LOGFILES	Specifies the number of logical-log files the database server is to create during disk initialization	page 37-34
SHMVIRTSIZE	Specifies the initial size of the virtual portion of shared memory	page 37-67
STACKSIZE	Specifies the stack size for Universal Server user threads	page 37-69

Setting Configuration Parameters for the Shared-Memory Performance Options with ON-Monitor

To set the configuration parameters for the shared-memory performance options using ON-Monitor, select Parameters from the main menu and then select the perFormance option. Figure 13-7 shows the perFormance screen. The shaded entries set the configuration parameters for the shared-memory performance options.

Figure 13-7
*ON-Monitor
PerFormance
Screen*

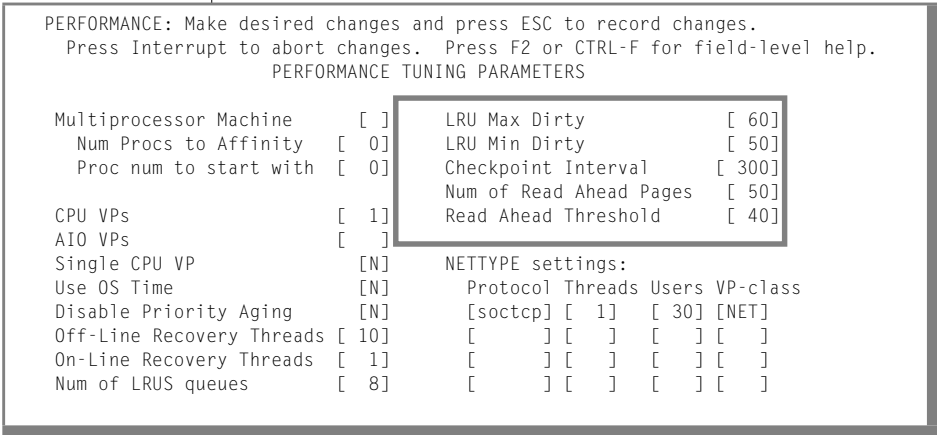
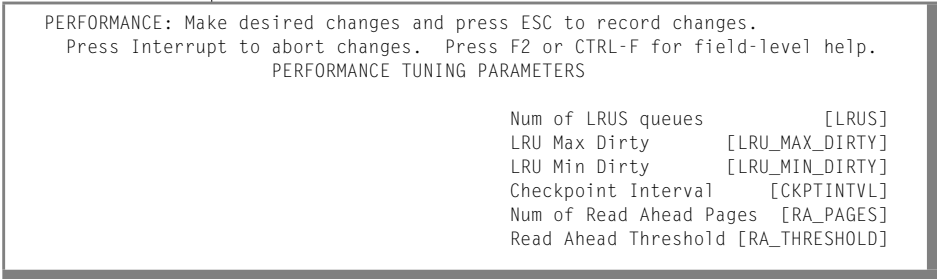


Figure 13-8 shows only the perFormance screen entries for setting the shared-memory performance options. For each entry, it shows the name of the associated parameter in the ONCONFIG file within brackets ([]).

Figure 13-8
*ONCONFIG
Parameters for the
Shared-Memory
Performance
Options*



For more information on the ONCONFIG parameters that set shared-memory performance options, refer to Figure 13-9.

Setting Configuration Parameters for the Shared-Memory Performance Options with a Text Editor

You can use a text editor program 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 13-9 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 37, “Configuration Parameters.”](#)

Figure 13-9
Setting Shared-Memory Performance Options

ONCONFIG Parameter	Purpose	Reference
CKPTINTVL	Specifies the maximum number of seconds that can elapse before Universal Server checks if a checkpoint is needed	page 37-11
LRU_MAX_DIRTY	Specifies the percentage of modified pages in the LRU queues that flags page cleaning to start	page 37-37
LRU_MIN_DIRTY	Specifies the percentage of modified pages in the LRU queues that flags page cleaning to stop	page 37-38
LRUS	Specifies the number of LRU queues for the shared-memory buffer pool	page 37-37
RA_PAGES	Specifies the number of disk pages that Universal Server should attempt to read ahead when it performs sequential scans of data or index records	page 37-58
RA_THRESHOLD	Specifies the number of unprocessed memory pages that, after they are read, cause Universal Server to read ahead on disk	page 37-59

Reinitializing Shared Memory

Universal Server reinitializes shared memory when you take Universal Server from off-line mode to quiescent mode or when you take it from off-line mode directly to on-line mode. So, to reinitialize shared memory, first bring Universal Server off-line. For information on how to take Universal Server from on-line mode to off-line, refer to [Chapter 8, “Managing Modes.”](#)

After Universal Server is off-line, you need to bring it to quiescent mode or on-line mode to reinitialize shared memory. For more information, refer to [“From Off-Line to Quiescent” on page 8-3](#) and [“From Off-Line to On-Line” on page 8-4](#).

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 Universal 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 Universal Server shared memory.

For a description of the resident portion of shared memory, refer to [“The Resident Portion of Database Server Shared Memory” on page 12-17](#).

Turning Residency On or Off in On-Line Mode

You can turn residency on or off while Universal 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 Universal 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 Universal Server terminates before the next checkpoint, the index build will restart the next time you initialize Universal Server.
- You are about to attempt a system operation which might risk power or otherwise interrupt Universal 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.

As user **informix**, you can use ON-Monitor to force a checkpoint by selecting the Force-Ckpt option from the main menu.

You can also force a checkpoint by executing the following command from the command line:

```
% onmode -c
```

Where Is Data Stored?

Overview of Data Storage	14-3
What Are the Physical Units of Storage?	14-5
What Is a Chunk?	14-5
Limits on Chunk Size and Number	14-6
Should You Allocate Chunks as Cooked Files or Raw Disk Space?	14-6
What Is an Offset?	14-9
What Is a Page?	14-10
What are Blobpages and Sbpages?	14-11
How Big Should Blobpage or Sbpages Be?	14-12
What Is an Extent?	14-13
What Are Disabling I/O Errors?	14-14
What Are the Logical Units of Storage?	14-16
What Is a Dbpace?	14-16
How Can You Control Where Data Is Stored?	14-17
How Does Table Fragmentation Affect Data Storage?	14-19
What Is the Root Dbpace?	14-21
What Is a Temporary Dbpace?	14-21
What Are the Advantages of Using Temporary Dbspaces?	14-22
What Is a Blobspace?	14-23
What Is an Sbspaces?	14-24
What Is Metadata and User Data?	14-24
What Is a Extspace?	14-25
What Is a Database?	14-25
What Is a Table?	14-26
What Is a Temporary Table?	14-28
Where Are Temporary Tables Stored?	14-29
What Is a Tblspace?	14-31
What Is Extent Interleaving?	14-32

How Much Disk Space Do You Need to Store Your Data?	14-33
Calculate the Size of the Root Dbspace	14-33
Physical and Logical Logs	14-33
Temporary Tables	14-34
Critical Data	14-34
ON-Archive Catalog Data	14-35
Control Information (Reserved Pages).	14-35
Complete the Root Dbspace Calculation	14-35
Estimate Space That Databases Require	14-36
Disk-Layout Guidelines	14-36
General Dbspace/Chunk Guidelines	14-37
Strive to Associate Partitions with Chunks	14-37
Mirror Critical Media	14-37
Spread Your Temporary Storage Space Across Multiple Disks	14-38
Move the Logical and Physical Logs from the Root Dbspace	14-38
Take into Account Backup-and-Restore Performance	14-39
Table-Location Guidelines	14-39
Isolate High-Access Tables.	14-40
Consider Mirroring	14-41
Group Your Tables with Backup and Restore in Mind	14-41
Place High-Access Tables on Middle Partition of Disk	14-42
Optimize Table-Extent Sizes	14-43
Sample Disk Layouts	14-43
Sample Layout When Performance Is Highest Priority	14-45
Sample Layout When Availability Is Highest Priority	14-47
What Is a Logical-Volume Manager?	14-49

T

his chapter defines terms and explains the concepts that you need to understand to perform the tasks described in [Chapter 15, “Managing Disk Space.”](#) This chapter covers the following topics:

- Definitions of the physical and logical units that INFORMIX-Universal Server uses to store data on disk
- Instructions on how to calculate the amount of disk space you need to store your data
- Guidelines on how to lay out your disk space and where to place your databases and tables

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 13](#) of the Introduction.

Overview of Data Storage

Universal Server can use two distinct types of disk space:

- Cooked file space, in which UNIX manages physical disk I/O
- Raw disk space, in which Universal Server manages physical disk I/O

Universal Server uses the following physical units to manage disk space:

- Chunk
- Page
- Blobpage
- Sbpage
- Extent

Overlying the physical units of storage space, Universal Server supports the following logical units associated with database management:

- Dbspace
- Blobspace
- Sbspace
- Extspace
- Database
- Table
- Tblspace

Universal 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 22, “What Is the Logical Log?”](#) For information about the physical log, see [Chapter 24, “What Is Physical Logging?”](#) For information about reserved pages, see [“Reserved Pages” on page 42-6.](#)

The following sections describe the various data-storage units that Universal Server supports and the relationships between those units.

What Are the Physical Units of Storage?

Universal 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, sbpages, and blobpages—has a fixed or assigned size.

What Is a Chunk?

The *chunk* is the largest unit of physical disk dedicated to Universal Server data storage. It represents an allocation of cooked disk space or raw disk space and is the *only* unit of physical storage that the Universal Server administrator allocates. The Universal Server administrator typically adds a chunk to a dbspace when that dbspace approaches full capacity. Figure 14-1 illustrates how disk space is allocated for a chunk on a raw device. The chunk is represented by the darkened concentric circles on the disk-drive platters. See “What Is a Raw Device?” on page 14-6.

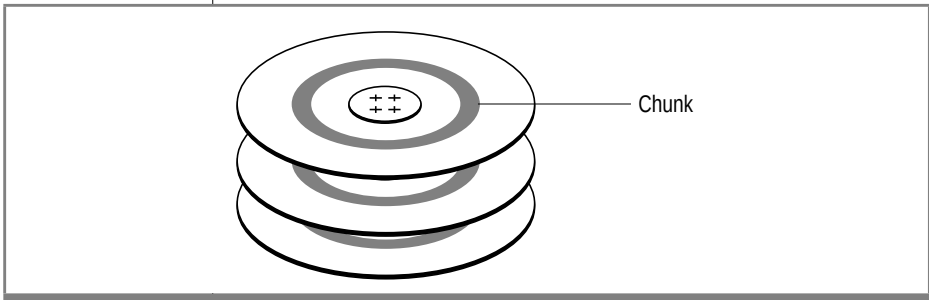


Figure 14-1
*A Typical Chunk
Allocated as Raw
Disk Space*

Chunks provide administrators with a conveniently large unit for allocating disk space. Universal Server also uses chunks for mirroring. A *primary chunk* is a chunk from which Universal Server copies data to a *mirrored chunk*. If the primary chunk fails, Universal Server brings the mirrored chunk on-line automatically. For more information on mirroring, see Chapter 27, “What Is Mirroring?”

Limits on Chunk Size and Number

On most platforms, the maximum size of a chunk is 2 gigabytes, although on some platforms the maximum chunk size is 4 gigabytes.

The maximum number of chunks that you can allocate for a given Universal Server system is 2048.

Should You Allocate Chunks as Cooked Files or Raw Disk Space?

This section describes the advantages and disadvantages of the two methods of allocating disk space: cooked files and raw disk space. As a general guideline, you experience better performance and increased reliability when you use raw disk space, but this outcome can vary depending on your operating system.

What Is a Raw Device?

The UNIX operating system uses the concept of a *device* to describe peripherals such as magnetic disks and tapes, terminals, and communication lines. One type of UNIX 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 *character-special* interface that leaves the buffering to the application. When you configure a block device with a character-special interface, the device is called a *raw device*, and the storage space that the device provides is called *raw disk space*.

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.

Space in a chunk of raw disk space is physically contiguous.

What Is a Cooked File?

A cooked file is a UNIX file. Although Universal Server manages the contents of cooked files, the UNIX 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 UNIX file, Universal Server manages the *internal* arrangement of data within the file. Never edit the contents of a cooked file managed by Universal Server directly; to do so puts the integrity of your data at risk.

How Does Universal Server Manage Data Differently When It Is Stored in a Cooked File Instead of a Raw Disk Device?

When the UNIX kernel reads from a cooked file, it reads the data from disk into the kernel buffer pool. Later, a second copy operation copies it from the kernel buffer to the location requested by the application. This means, for example, that when two users both read the password file, the data is only read from disk once but copied from the kernel buffer twice.

By contrast, when the kernel reads data from a raw disk device, it bypasses the kernel buffer pool and copies the data directly to the location requested by the application. Universal Server requests that the data be placed in shared memory, making it immediately available to all Universal Server virtual processors and running threads with no further copying.

Why Use a Raw Device?

The character-special 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 to store your data, Universal 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 page 15-5](#).

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 Universal 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 very convenient.

When performance is not a consideration, you could also consider using 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 UNIX 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 UNIX 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 UNIX file manager instead of being written immediately to disk. As a consequence, Universal Server cannot guarantee that the committed data actually reaches the disk. Universal 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 Universal 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 Cooked File Space” on page 15-5](#).

What Is an Offset?

Although Informix recommends that you use an entire UNIX partition when you allocate a chunk (see [“Strive to Associate Partitions with Chunks” on page 14-37](#) for more information), you can subdivide partitions or cooked files into smaller chunks using *offsets*.

From the perspective of the UNIX operating system, a chunk is a stream of bytes. An offset allows you to indicate the number of kilobytes into a raw 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 wish 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 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 Universal Server from overwriting partition information. [“Do You Need to Specify an Offset?” on page 15-6](#) explains when and how (using ON-Monitor or **onspaces**) to specify an offset.

What Is a Page?

A *page* is the physical unit of disk storage that Universal 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. (See [“Determining Universal Server Page Size” on page 15-15.](#)) Because your hardware determines the size of your page, you cannot alter this value. Figure 14-2 illustrates the concept of a page, represented by a darkened sector of a disk platter.

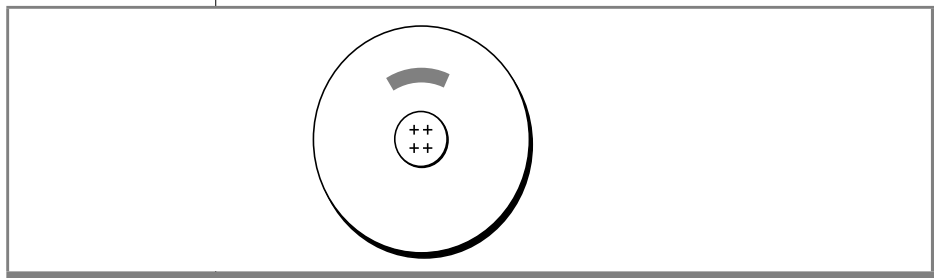


Figure 14-2
A Page on Disk

A chunk contains a certain number of pages, as illustrated in Figure 14-3. A page is always entirely contained within a chunk; that is, a page cannot cross chunk boundaries.

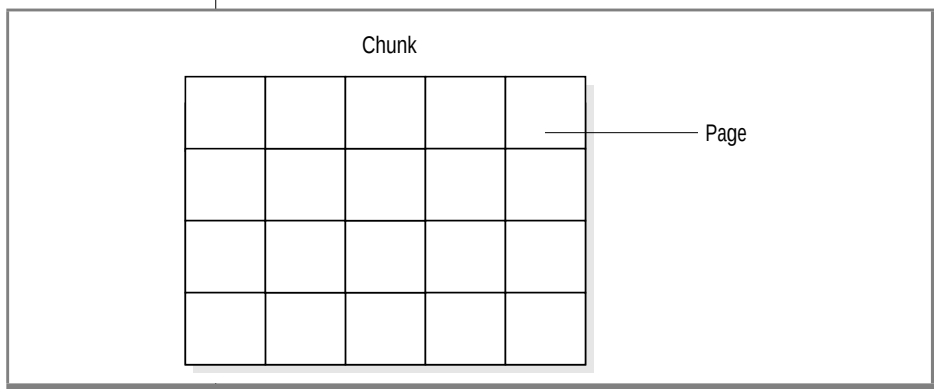


Figure 14-3
A Chunk, Logically Separated into a Series of Pages

For information on how Universal Server structures data within a page, see [Chapter 42, “Disk Structures and Storage.”](#)

What are Blobpages and Sbpages?

A blobpage is the unit of disk-space allocation that Universal Server uses to store simple large objects within a blobspace, while a sbpage is the unit of disk-space allocation that Universal Server uses to store smart large objects within a sbpace. (See [“What Is a Blobspace?”](#) on page 14-23 and [“What Is an Sbpace?”](#) on page 14-24.) You specify blobpage and sbpage size as a multiple of the Universal Server page size. (See [“Optimizing Blobspace Blobpage Size”](#) on page 15-21.) The Universal Server administrator establishes the size of a blobpage when creating the blobspace in which that blobpage resides; the size of a blobpage can vary from blobspace to blobspace. Figure 14-4 illustrates the concept of a blobpage, represented as a multiple (three) of a data page (the concepts for sbpages are identical).

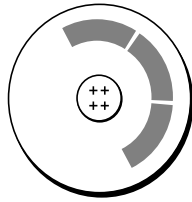


Figure 14-4
A Blobpage on Disk

Just as with pages in a chunk, a certain number of blobpages compose a chunk in a blobspace, as illustrated in Figure 14-5. A blobpage is always entirely contained in a chunk and cannot cross chunk boundaries.

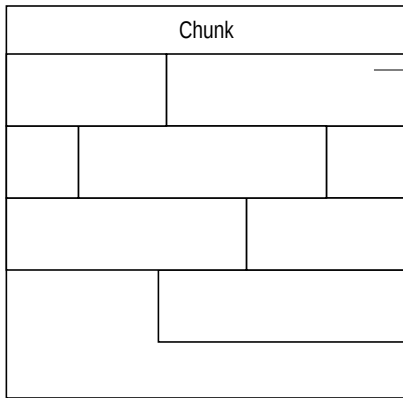


Figure 14-5
*A Chunk in a
Blobspace, Logically
Separated into a
Series of Blobpages*

What are Blobpages and Sbpages?

In addition to storing your simple-large-object data in a blobpage, you can also choose to store your simple-large-object data in a dbpage. However, for simple large objects larger than two pages, performance improves when you store the simple large objects in a blobpage. simple large objects stored in a dbpage can share a page; simple large objects stored in a blobpage do not share pages. smart large objects cannot be stored in a dbpage. For information about how Universal Server structures data stored in a blobpage, see [“Structure of a Blobpage” on page 42-68](#).

How Big Should Blobpage or Sbpages Be?

When you create a blobpage or sbpage, try to create a blobpage or sbpage size that approximates the size of the most frequently occurring simple or smart large object that the blobpage or sbpage holds. For example, if you are storing 160 simple large objects, and you expect 120 simple large objects to be 12 kilobytes and 40 simple large objects to be 16 kilobytes, a 12-kilobyte blobpage size stores the simple large objects most efficiently. This configuration allows the majority (120) of the simple large objects to be stored in a single blobpage, while the other 40 simple large 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 does not waste as much space as using a blobpage or sbpage 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 simple large 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 Universal 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 Universal Server page size is 4 kilobytes, specify the blobpage size parameter as 3 (pages). In general, divide the size of the simple large object (rounded up to the nearest kilobyte) by the page size to determine the blobpage size parameter.

If a table has more than one simple-large-object column, and the simple large objects are not close in size, store the simple large objects in different blobspaces, each with an appropriately sized blobpage. See [“What Is a Table?”](#) on page 14-26.

What Is an Extent?

When you create a table, Universal Server allocates a fixed amount of space to contain the data to be stored in that table. When this space fills, Universal Server must allocate space for additional storage. The physical unit of storage space that Universal Server uses to allocate both the initial and subsequent storage space is called an *extent*. Figure 14-6 illustrates the concept of an extent.

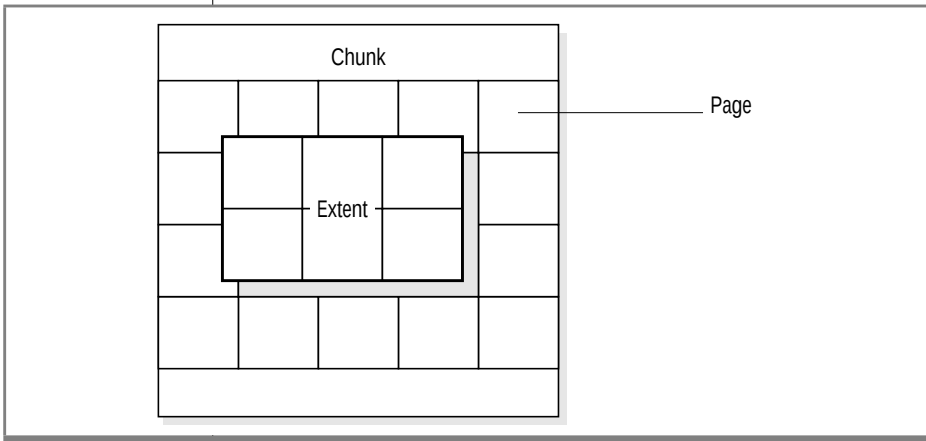


Figure 14-6
An Extent That
Consists of Six
Contiguous Pages
on a Raw Disk
Device

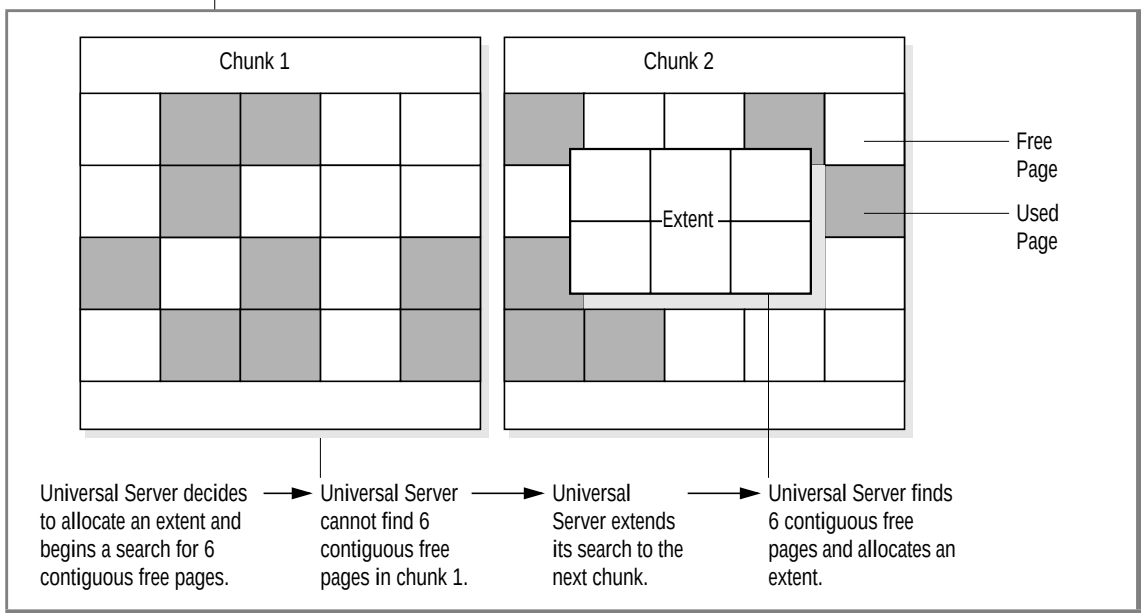
An extent consists of a collection of contiguous pages that store data for a given table. (See [“What Is a Table?”](#) on page 14-26.) Every permanent database table has two extent sizes associated with it. The *initial-extent* size is the number of kilobytes allocated to the table when it is first created. 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](#).

What Is an Extent?

Figure 14-7 illustrates the following key concepts concerning extent allocation:

- An extent is always entirely contained in a chunk; an extent cannot cross chunk boundaries.
- If Universal 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 14-7
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 Universal Server marks the chunk and dbspace as down. Universal 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 Universal Server considers an I/O error to be disabling, the error must meet two criteria. First, the error must occur when Universal 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 Universal 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

Universal Server performs this verification as a sanity check immediately after it opens a chunk.

You can prevent Universal 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 Universal 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 Universal Server storage fall into the following categories:

- Units of logical storage that function as accounting entities, including:
 - ❑ Dbspaces
 - ❑ Blobspace
 - ❑ Sbspace
 - ❑ Extspace
 - ❑ 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 14-26.](#))

The following sections describe these logical storage units.

What Is a Dbspace?

A key responsibility of the Universal Server administrator is to control where Universal Server stores data. By storing high-access tables or critical media (root dbspace, physical log, and logical log) on your fastest disk drive, you can improve performance. By storing critical media and data on separate physical devices, you ensure that when one of the disks holding noncritical media fails, the failure affects only the availability of data on that disk.

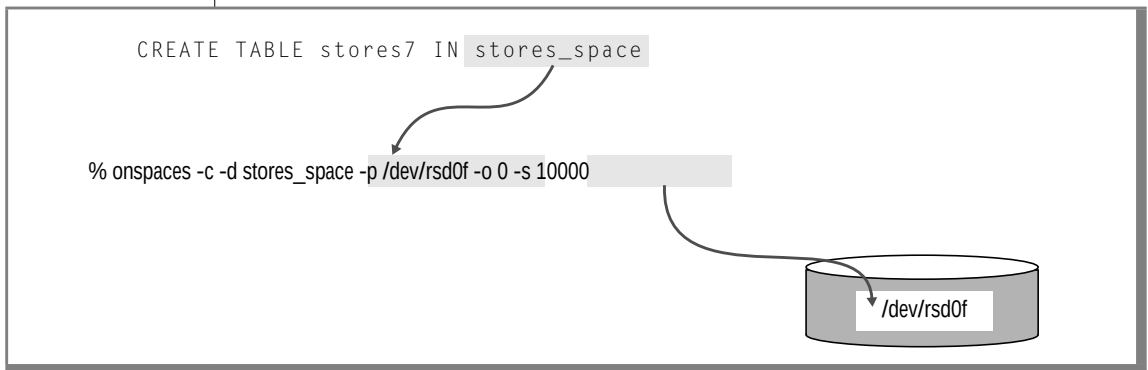
These strategies require the ability to control the location of data. The logical storage unit that provides this ability is the *dbspace*. The dbspace provides the critical link between the logical and physical units of storage. It allows you to associate physical units (such as chunks) with logical units (such as tables).

How Can You Control Where Data Is Stored?

As Figure 14-8 shows, you control the placement of databases or tables (see “What Is a Table?” on page 14-26) using the IN *dbspace* option of the CREATE DATABASE or CREATE TABLE statements.

Figure 14-8

Controlling Table Placement with the CREATE TABLE... IN Statement



Before you create a database or table in a dbspace, you must first create the dbspace using the **onspaces** utility or ON-Monitor. For more information on how to create a dbspace, see “Creating a Dbspace” on page 15-9.

What Is a Dbspace?

A dbspace includes one or more chunks, as shown in Figure 14-9. You can add more chunks at any time. As with blobspace chunks, it is a high-priority task of an Universal 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 33-58.](#)) When a dbspace contains more than one chunk, you cannot specify the chunk in which the data resides.

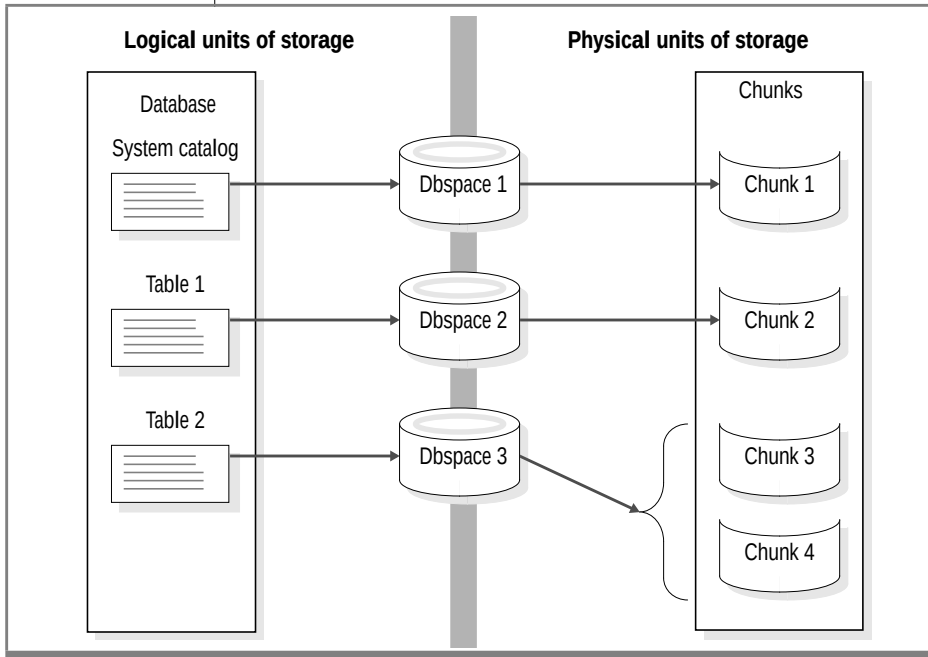


Figure 14-9
*Dbspaces That Link
Logical and Physical
Units of Storage*

Universal Server uses the dbspace to store databases and tables. (See [“What Is a Table?” on page 14-26.](#)) You can store BYTE and TEXT data within a dbspace, but if the simple large objects are larger than two pages, performance might be worse than if you stored the same data in a blobspace.

You must mirror every chunk in a mirrored dbspace. As soon as Universal Server allocates a mirrored chunk, it flags all space in that mirrored chunk as full. See [“Monitoring Chunk Status” on page 33-55.](#)

You can use ON-Monitor or **onspaces** to perform any of the following tasks related to dbspace management:

- Creating a dbspace ([page 15-9](#))
- Adding a chunk to a dbspace ([page 15-12](#))
- Dropping a dbspace or blobspace ([page 15-19](#))

You can use **onspaces** (but not ON-Monitor) to drop a chunk from a dbspace. For more information, see “[Dropping a Chunk from a Dbspace with onspaces](#)” on [page 15-18](#).

How Does Table Fragmentation Affect Data Storage?

The fragmentation feature gives you additional control over where Universal Server 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*.

With fragmentation, you can define groups of rows within a table according to some *distribution scheme* and specify a different dbspace for each of these groups. You can also define a distribution scheme for indexes. You can specify that the index fragments are stored in the same dbspaces as their corresponding table fragments or in other, unrelated dbspaces.

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 contained in fragmented tables.

From the perspective of Universal Server, however, a fragmented table and its fragmentation strategy are apparent. Depending on how you fragment your tables, Universal Server can route client requests for data to the appropriate fragment without accessing irrelevant fragments.

What Is a Dbspace?

Figure 14-10 illustrates the role of fragments in specifying the location of data.

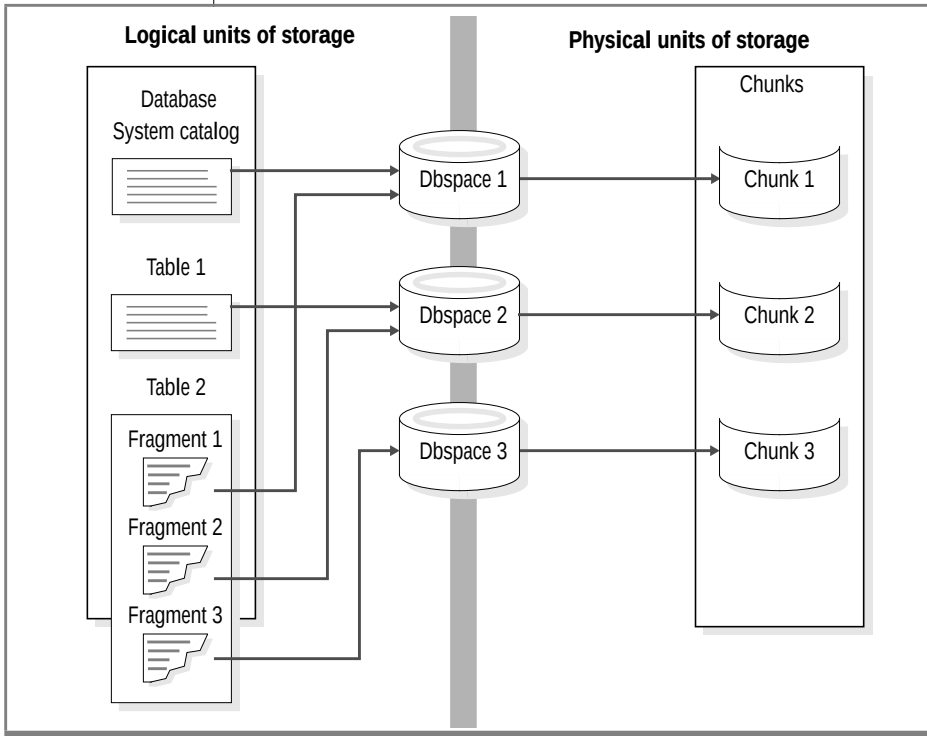


Figure 14-10
*Dbspaces That Link
Logical Units
(Including Table
Fragments) and
Physical Units of
Storage*

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, refer to [Chapter 16, “What Is Fragmentation?”](#)

What Is the Root Dbspace?

The root dbspace is the initial dbspace that Universal Server creates. The root dbspace is special because it contains reserved pages and internal tables (see [“What Is a Table?” on page 14-26](#)) that describe and track all other dbspaces, blobspaces, sbspaces, extspaces, chunks, databases, and tblspaces. (For more information on these topics, see [Chapter 42, “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 Universal 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 14-33](#) explains how much space to allocate for the root dbspace. You can also add extra chunks to the root dbspace after you initialize Universal 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 14-28](#).)

Universal Server never drops a temporary dbspace unless it is explicitly directed to do so. A temporary dbspace is only temporary in the sense that Universal Server does not preserve any of the dbspace contents when Universal Server shuts down abnormally. Temporary dbspaces are designed exclusively for the storage of temporary tables.

Whenever you initialize Universal Server, all temporary dbspaces are reinitialized. Universal Server clears any tables that might be left over from the last time that Universal Server shut down.

Universal Server does not perform logical or physical logging for temporary dbspaces. Neither ON-Archive nor **ontape** includes 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 with ON-Monitor or **onspaces**, see [“Creating a Dbspace” on page 15-9](#).



Important: When Universal Server is running as a secondary database server in a 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?

Universal Server logs table creation, the allocation of extents, and the dropping of the table for a temporary table in a standard dbspace. In contrast, Universal 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.

Universal 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, Universal 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 Universal Server does not backup temporary dbspaces.

What Is a Blobospace?

A blobospace is a logical storage unit composed of one or more chunks that store only BYTE and TEXT data, also called *simple large objects*. A blobospace stores BYTE and TEXT data in the most efficient way possible. You can store simple large objects associated with distinct tables (see [“What Is a Table?” on page 14-26](#)) in the same blobospace.

Universal Server writes simple-large-object data stored in a blobospace 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, Universal Server does not write blobospace simple large objects to either the logical or physical log. Universal Server logs blobospace simple large objects by writing the simple large objects directly from disk to the logical-log backup tapes when you back up the logical logs. Blobospace simple large objects never pass through the logical-log files.

When you create a blobospace, you assign to it one or more chunks. You can add more chunks at any time. One of the tasks of an Universal Server administrator is to monitor the chunks for fullness and anticipate the need to allocate more chunks to a blobospace. For instructions on how to monitor chunks for fullness, see [“Monitoring Simple Large Objects in a Blobospace” on page 33-69](#). For instructions on how to create a blobospace, add chunks to a blobospace, or drop a chunk from a blobospace, see [Chapter 15, “Managing Disk Space.”](#)

For information about the structure of a blobospace, see [“Structure of a Blobospace” on page 42-62](#).

What Is an Sbspace?

A sbspace is a logical storage unit composed of one or more chunks that store only CLOB (character large object) and BLOB (binary large object) data, also called *smart large objects*. Unlike simple large objects, smart large objects have read, write, and seek properties similar to a standard UNIX file. In addition, smart large objects are recoverable and obey transaction isolation modes. Also, smart large objects within table rows do not need to be retrieved in one statement; an application can store or retrieve smart large objects in pieces. CLOB smart large objects can contain only text data, including formatted text such as HTML or PostScript. BLOB smart large objects can contain any type of data.

As with blobspaces and dbspaces, when you create a sbspace, you assign to it one or more chunks. However, when you add a chunk to an sbspace, you can specify whether it contains *metadata* partitions and *user data* partitions or whether it is reserved exclusively for user data or metadata (See “What Is Metadata and User Data?” in the following section). You can add more chunks at any time. For instructions on how to create a sbspace, add chunks to a sbspace, or drop a chunk from a sbspace, see [Chapter 15, “Managing Disk Space.”](#)

What Is Metadata and User Data?

Metadata identifies key aspects of the sbspace and each smart large object stored in the sbspace, and enables Universal Server to manipulate and recover smart large objects stored within. User data is the data stored in the sbspace by user applications.

For information on correctly allocating metadata and user data for sbspaces, see “[Sizing Sbspace Metadata](#)” on page 15-17.

Important: *Sbspace metadata is always logged, regardless of the logging setting of the database.*



What Is a Extspace?

An extspace is simply a logical name associated with an arbitrary string that can point to any type of file on any type of device. The resource referenced by the extspace depends upon a user-defined access method for accessing its contents.

For example, a database user might require access to a binary files encoded in a proprietary format. First, a developer codes a routine to access the file. This routine is responsible for all interaction between Universal Server and the external file. A DBA then adds an extspace that has the file as its target to the database and creates access method functions and a primary access method. Users can now access the data in the proprietary files via SQL statements. The access method would locate those files by using the extspace information.

For more information on user-defined access methods, see the *Virtual-Table Interface Programmer's Manual*. For more information on creating functions and primary access methods, see the [Informix Guide to SQL: Syntax](#).

What Is a Database?

A database is a logical storage unit that contains tables (see “[What Is a Table?](#)” on page 14-26) 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 14-11 on page 14-26](#) 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

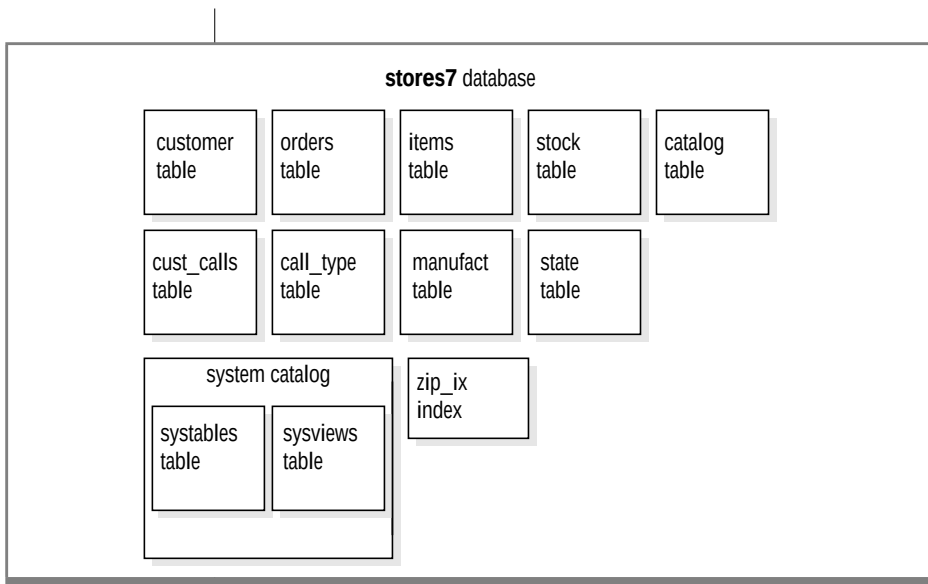


Figure 14-11
*The stores7
Database*

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 33-47](#).

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, Universal Server allocates disk space for the table in a block of pages called an extent. (See [“What Is an Extent?” on page 14-13](#).) 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, Universal 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 dbspace. (For more information, see [Chapter 16, “What Is Fragmentation?”](#))

A table or table fragment resides completely in the dbspace in which it was created. The Universal 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 14-12 shows.

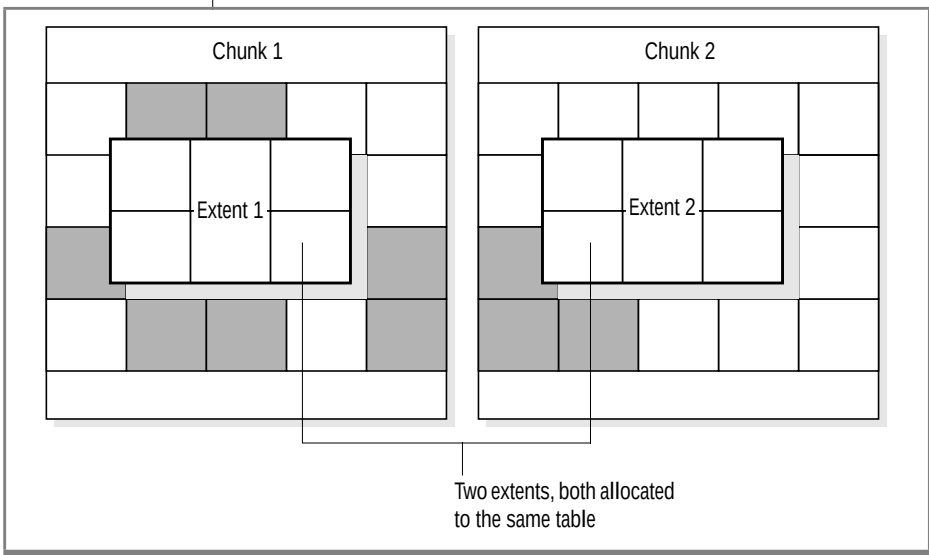


Figure 14-12
*Table That Spans
More than One
Chunk*

Simple-large-object data associated with a table can reside either in the dbspace with the rest of the table data or in a separate blobspace. When you use INFORMIX-OnLine/Optical, you can also store simple large objects in an optical storage subsystem.

For advice on where to store your tables, see “[Disk-Layout Guidelines](#)” on [page 14-36](#) and also Chapter 3 of the *INFORMIX-Universal Server 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, Universal 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, Universal Server deletes the temporary table.

An *implicit* temporary table is a temporary table that Universal 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

Universal Server deletes an implicit temporary table when the processing that initiated the creation of the table is complete.

If Universal 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.)



Important: *Temporary tables are not supported for external tables.*

Where Are Temporary Tables Stored?

The dbspace in which Universal 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, Universal 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, Universal Server checks the **DBSPACETEMP** environment variable and the DBSPACETEMP configuration parameter. (The environment variable supersedes the configuration parameter.) When **DBSPACETEMP** is set, Universal Server stores the explicit temporary table in one of the dbspaces specified in the list.

What Is a Temporary Table?

Universal Server keeps track of the last dbspace in the list that it used to store a temporary table. When Universal Server receives another request for temporary storage space, it uses the next dbspace in the list. In this way, Universal 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, Universal Server creates the table in a standard (nontemporary) dbspace according to the following rules:

- If you created the temporary table with **CREATE TEMP TABLE**, Universal 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, Universal Server stores this table in the root dbspace.

Implicit Temporary Tables

Universal 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, Universal Server stores the temporary table in the root dbspace.

When Universal 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 the [INFORMIX-Universal Server Performance Guide](#).

What Is a Tblspace?

Universal 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 simple-large-object data in the *dbspace* (but not pages used to store simple-large-object data in a *blobspace*)
- Bit-map pages that track page use within the table extents

Figure 14-13 illustrates the *tblspaces* for three tables that form part of the **stores7** database. Only one table (or table fragment) exists per *tblspace*. Simple-large-object pages represent simple-large-object data stored in a *dbspace*.

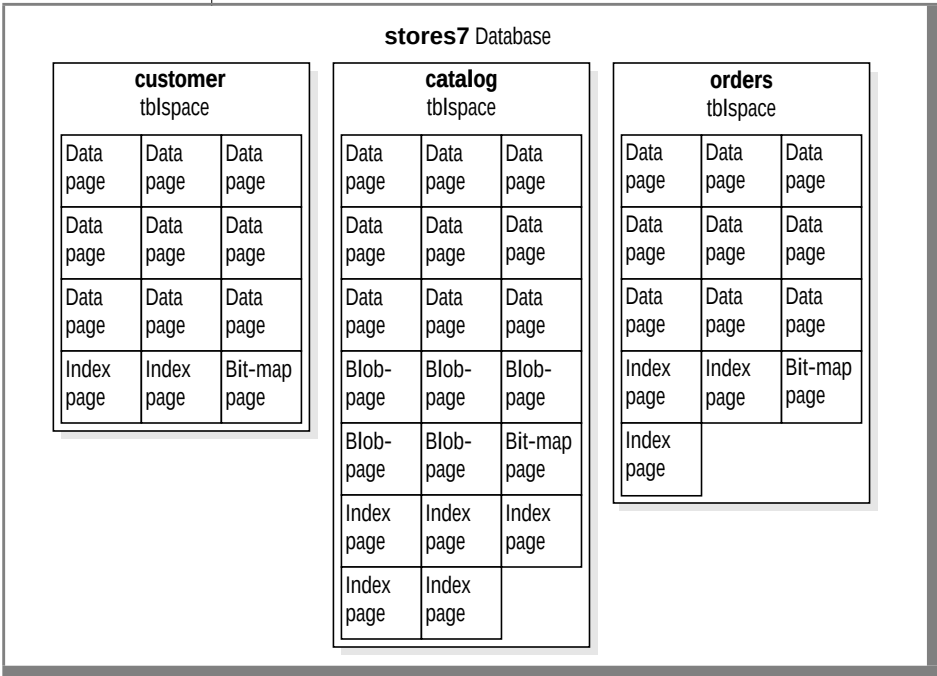


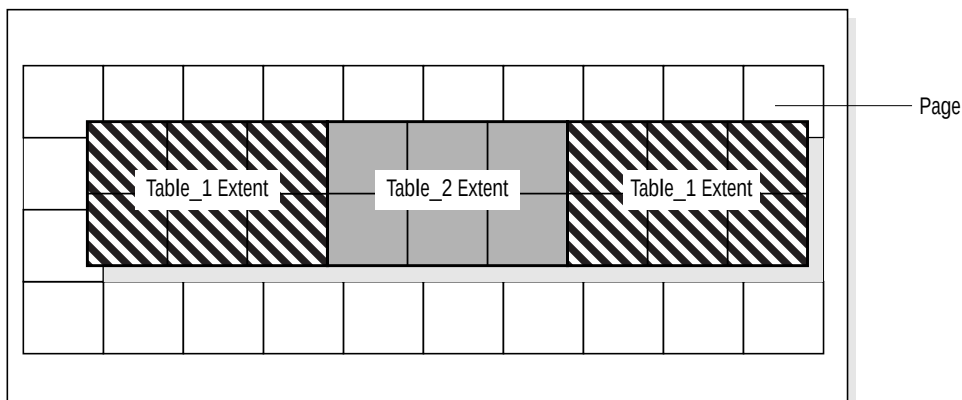
Figure 14-13
*Three of the Sample
Tblspaces in the
stores7 Database*

What Is Extent Interleaving?

Universal 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 14-14 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 Chapter 3 of the [INFORMIX-Universal Server Performance Guide](#).

Figure 14-14

Three Extents That Belong to Two Different Tblspaces 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 Universal 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)

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 24-5](#).

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 22-7](#).

Temporary Tables

Analyze end-user applications to estimate the amount of disk space that Universal Server might require for implicit temporary tables. [“What Is a Temporary Table?” on page 14-28](#) 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.

Universal Server creates implicit temporary files when you use ON-Archive to perform a warm restore. The largest implicit temporary file that Universal 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 22-12](#).

You must also analyze end-user applications to estimate the amount of disk space that Universal Server might require for explicit temporary tables. See [“What Is a Temporary Table?” on page 14-28](#).

By default, Universal 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 14-29](#).

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.

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 the [INFORMIX-Universal Server Archive and Backup Guide](#). For instructions on how to calculate the size of each table, see the [INFORMIX-Universal Server Performance Guide](#).

Control Information (Reserved Pages)

The total amount of disk space required for Universal 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 Universal 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
- 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 Universal 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 Chapter 3 of the [*INFORMIX-Universal Server Performance Guide*](#).

Disk-Layout Guidelines

The following are typical goals for efficient disk layout:

- Limiting disk-head movement
- Reducing disk contention
- Balancing the load
- Maximizing availability

You must make some trade-offs 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.

General Dbospace/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 (raw disk or cooked files), you allocate it in chunks. A dbospace or a blobspace is associated with one or more chunks. You must allocate at least one chunk for the root dbospace.

Informix recommends that you format your disks so that each chunk is associated with its own UNIX 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 [“Do You Need to Specify an Offset?” on page 15-6](#).

Mirror Critical Media

Mirror the critical media: the root dbospace, the dbospace that contains the physical log, and the dbospace 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 14-15 shows a primary chunk and its mirror.

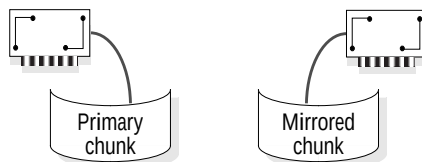


Figure 14-15
*Ideal Disk Layout for
Primary Chunk and
Associated Mirrored
Chunk*

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 37-16](#).

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 Universal 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, Universal 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-access 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 24-8](#) and [“Where Should Logical-Log Files Be Located?” on page 22-13](#).

For instructions on how to change the location of the logical and physical log, see [“Changing the Physical-Log Location and Size” on page 25-3](#) and [“Moving a Logical-Log File to Another Dbspace” on page 23-8](#).

Take into 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 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, Universal Server comes off-line. In addition, the Universal Server administrator must restore all Universal 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).

[“Fragmenting for Improved Backup-and-Restore Characteristics” on page 16-28](#) explains this concept in detail.

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 [“Formulating a Fragmentation Strategy” on page 16-23](#).

Isolate High-Access 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-access 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-access 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 14-16 illustrates this strategy by showing optimal placement of three frequently accessed tables.

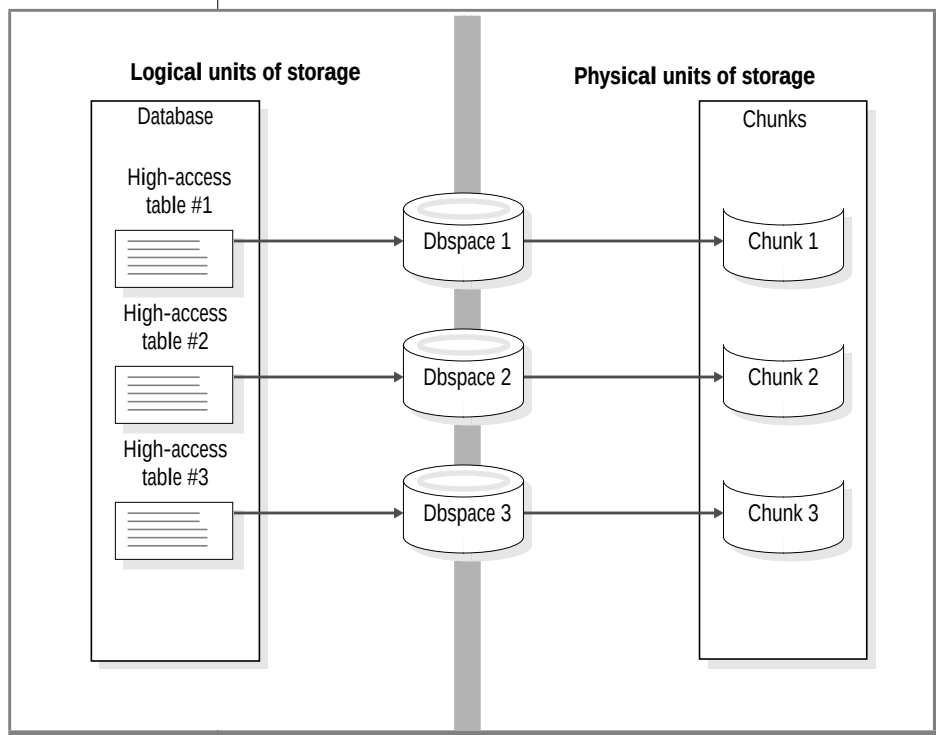


Figure 14-16
Example of High-Access Table Isolation

To take this strategy a step further, fragment a high-access table over multiple disk devices. If you choose an appropriate distribution scheme, Universal Server routes queries to the appropriate fragment, thereby reducing contention on any single fragment. For more information, see [“Fragmenting for Improved Concurrency” on page 16-26](#).

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 39-88](#).

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. See [“Fragmenting for Improved Availability of Data” on page 16-27](#).

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 the [INFORMIX-Universal Server Backup and Restore Guide](#) for more information. If you use ON-Archive or **ontape** for your backup and restore tool, refer to [INFORMIX-Universal Server 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. See [“Fragmenting for Improved Backup-and-Restore Characteristics”](#) on page 16-28.

Place High-Access 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 14-17. 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.

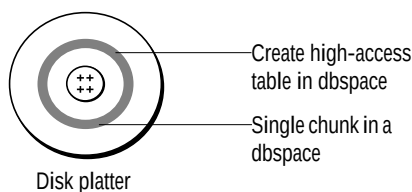


Figure 14-17
*Disk Platter with
High-Access Table
Located on Middle
Partitions*

To place high-access tables on the middle partition of the disk, create a raw device (see your UNIX manual for instructions on how to create a raw device) composed of cylinders that reside midway between the spindle and the outer edge of the disk. Allocate a chunk, associating it to this raw device. Then create a dbspace with this same chunk as the initial and only chunk. When you create your high-access tables, use the IN option of CREATE TABLE to place them in the newly created dbspace.

Optimize Table-Extent Sizes

As explained in “[What Is a Tblspace?](#)” on page 14-31, when two or more large, growing tables share a dbspace, their new extents can become interleaved. This interleaving creates gaps between the extents of any one table. (See [Figure 14-14 on page 14-32](#).) 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 Chapter 3 in the *[INFORMIX-Universal Server Performance Guide](#)*. You can also consider placing the tables in separate dbspaces.

Sample Disk Layouts

When setting out to organize disk space, a Universal 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 Universal 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.

Sample Disk Layouts

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 Universal Server 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-access 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 Universal Server uses to look up data such as zip-code or manufacturer.

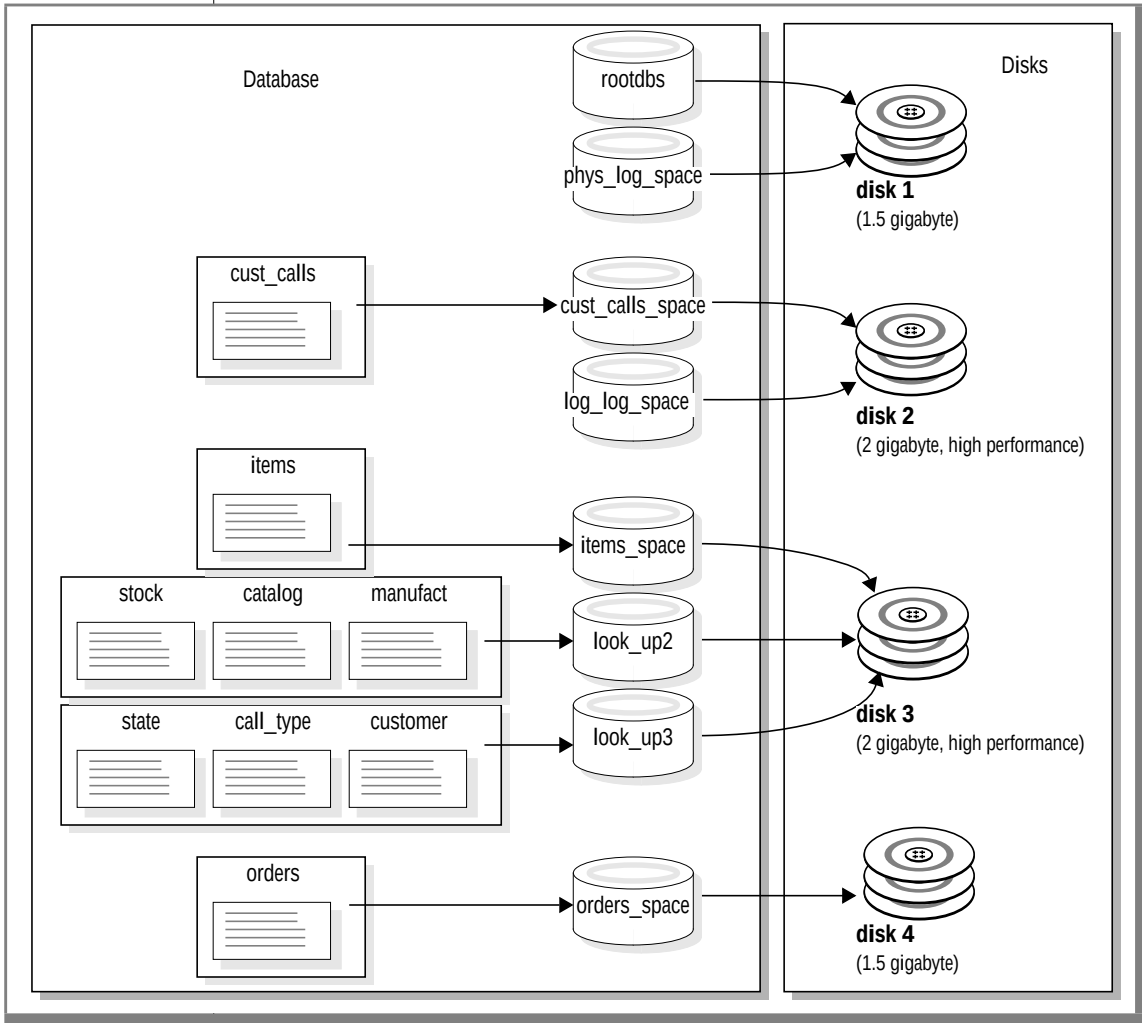
Table Name	Maximum Size	Access Rate
cust_calls	1.5 gigabytes	Low
items	0.5 gigabytes	High
orders	50 megabytes	High
customers	50 megabytes	Low
stock	50 megabytes	Low
catalog	50 megabytes	Low
manufact	50 megabytes	Low
state	50 megabytes	Low
call_type	50 megabytes	Low

Sample Layout When Performance Is Highest Priority

Figure 14-18 on page 14-46 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 14-18
Disk Layout Optimized for Performance

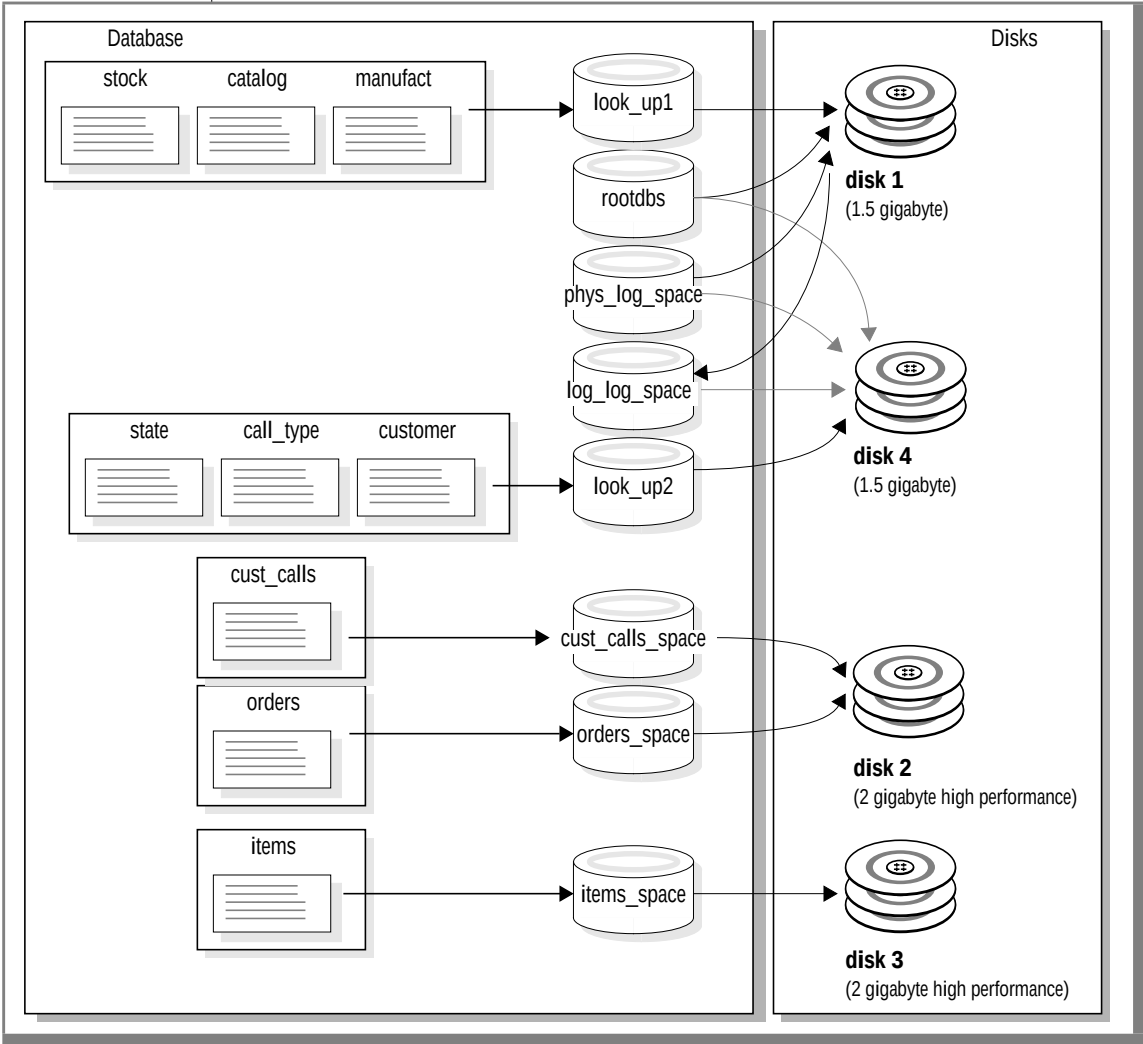


Sample Layout When Availability Is Highest Priority

The weakness of the previous disk layout is that if either disk 1 or disk 2 fails, the whole database server goes down until you restore the dbspaces on these disks from backups. In other words, the disk layout is poor with respect to availability.

An alternative disk layout that optimizes for availability is shown in [Figure 14-18 on page 14-46](#). This layout mirrors all the critical media (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 14-19
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 Universal 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. Universal 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-access tables in that volume. For more information, see [“Place High-Access Tables on Middle Partition of Disk” on page 14-42](#). (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 14-17](#).)

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-Access Tables on Middle Partition of Disk” on page 14-42](#).

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 27-6](#).

Managing Disk Space

Allocating Disk Space	15-4
Allocating Cooked File Space	15-5
Allocating Raw Disk Space.	15-5
Do You Need to Specify an Offset?.	15-6
Creating Links to Each Raw Device	15-7
Initializing Disk Space.	15-8
Initializing Disk Space with ON-Monitor	15-8
Initializing Disk Space with oninit	15-9
Creating a Dbospace.	15-9
Creating a Dbospace with ON-Monitor	15-10
Creating a Dbospace with onspaces	15-11
Adding a Chunk to a Dbospace	15-12
Adding a Chunk	15-12
Adding a Chunk with ON-Monitor	15-13
Adding a Chunk Using onspaces	15-13
Creating a Blobospace	15-14
Determining Universal Server Page Size	15-15
Creating a Blobospace with ON-Monitor	15-15
Creating a Blobospace with onspaces.	15-16
Adding a Chunk to a Blobospace	15-16
Creating a Sbspace	15-16
Sizing Sbspace Metadata	15-17

Adding a Chunk to a Sbspace	15-18
Dropping a Chunk from a Dbspace with onspaces	15-18
Dropping a Chunk from a Blobspace or Sbspace	15-19
Dropping a Dbspace, Blobspace, or Sbspace	15-19
Dropping a Dbspace or Blobspace with ON-Monitor	15-20
Dropping a Dbspace, Blobspace, or Sbspace with onspaces.	15-20
Optimizing Blobspace Blobpage Size	15-21
Determining Blobspace Storage Efficiency	15-21
Blobspace Storage Statistics.	15-22
Determining Blobpage Fullness with oncheck -pB	15-22
Interpreting Blobpage Average Fullness	15-24
Apply Efficiency Criteria to Output	15-25
Managing Extspaces	15-25
Creating a Extspace	15-25
Dropping a Extspace	15-26

T

his chapter provides the instructions that you need to manage effectively the disk space and data that INFORMIX-Universal Server controls. It assumes you are familiar with the terms and concepts contained in [Chapter 14, “Where Is Data Stored?”](#)

This chapter covers the following topics:

- How to allocate raw or cooked file space
- How to initialize disk space
- How to set configuration variables related to disk management
- How to manage blobspaces, sbspaces, chunks, and dbspaces
 - Allocating, adding, and dropping chunks from dbspaces, sbspaces, and blobspaces
 - Creating and dropping dbspaces, blobspaces, and sbspaces
- How to optimize blobpage size
- How to manage extspaces
 - Creating and dropping extspaces

The [INFORMIX-Universal Server Performance Guide](#) also contains information about managing disk space. In particular, Chapter 3 of that manual explains 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 Universal Server. Read the following sections before you allocate disk space:

- [“Should You Allocate Chunks as Cooked Files or Raw Disk Space?” on page 14-6](#)
- [“How Much Disk Space Do You Need to Store Your Data?” on page 14-33](#)
- [“Disk-Layout Guidelines” on page 14-36](#)

After you allocate the necessary space, you might still need to take additional steps before Universal 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, blobspace, or sbospace
- Adding a chunk to an existing dbspace, blobspace, or sbospace
- Mirroring an existing dbspace, blobspace, or sbospace

You can allocate disk space as raw disk space or as a cooked file. Informix recommends that, if you allocate raw disk space, you use the UNIX `link` command to create a link between the character-special device name and another filename. For more information on this topic, see [“Creating Links to Each Raw Device” on page 15-7](#).

Allocating Cooked File Space

To allocate cooked file space, log in as user **informix** and concatenate null to a pathname that represents one chunk of cooked file space. The cooked disk-space file should have permissions set to 660 (rw-rw----). Group and owner must be set to **informix**. Figure 15-1 illustrates these steps and assumes that you will store the cooked space in the file **/usr/data/my_chunk**.

Figure 15-1
Preparing Cooked File Space for Universal Server

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 <code>ls -l</code> 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).

Allocating Raw Disk Space

To allocate raw disk space, consult your UNIX system documentation for instructions on how to create and install a raw device.

In general, to create a raw device (see [“What Is a Raw Device?” on page 14-6](#)), 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.



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* (for example, **/dev/rsd0f**).

Verify that the UNIX permissions on the character-special devices are 660. Usually, the character-special designation and device permissions appear as `crw-rw----` if you execute the UNIX **ls -l** command on the filename. (Some UNIX systems might vary.)

Warning: After you create the raw device that Universal Server will use for disk space, carefully heed the following warnings:

- Do not create file systems on the same raw device that you allocate for Universal Server disk space.
- Do not use the same raw device as swap space that you allocate for Universal Server disk space.

Do You Need to Specify an Offset?

You can use offsets for two purposes:

- To prevent Universal Server from overwriting the UNIX partition information
- To define multiple chunks on a partition, disk device, or cooked file

Both uses are described in this section.

Many UNIX 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 Universal Server from overwriting critical information required by UNIX. For the exact offset required, refer to your disk-drive manuals.

You can also use offsets to define multiple chunks on a partition, disk device, or cooked file. You define the chunks by specifying a beginning offset when you add a chunk to a dbspace or blobspace with **onspaces** or ON-Monitor. The offset parameter specifies the beginning byte of the chunk. Universal Server determines the last byte of the chunk by adding the chunk size in bytes to the beginning offset.

For the initial chunk of root dbspace or its mirror, specify an offset with the ROOTOFFSET and MIRROROFFSET parameters, respectively. For the initial chunk of nonroot dbspaces, you supply the offset as a parameter when you create the dbspace with **onspaces** or ON-Monitor. See [“Creating a Dbspace” on page 15-9](#).



Warning: *If you are running two or more instances of Universal Server, be extremely careful not to define chunks that overlap. Overlapping chunks can cause Universal Server to overwrite data in one chunk with unrelated data from an overlapping chunk. This overwrite effectively destroys overlapping data.*

Creating Links to Each Raw Device

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 Universal 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
```

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 Universal Server database server. This action occurs when you bring Universal Server on-line for the first time.



Warning: When you initialize Universal Server disk space, you overwrite whatever is on that disk space. If you reinitialize disk space for an existing Universal Server database server, all data in the earlier Universal Server instance becomes inaccessible and, in effect, is destroyed.

To execute the **oninit** process, enter the command **oninit** (with or without command-line options) at the UNIX prompt or request initialization through ON-Monitor.

Only user **informix** or **root** can execute **oninit** and initialize Universal Server. Universal 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**.

Initializing Disk Space with ON-Monitor

To initialize Universal Server with ON-Monitor, select the Parameters menu, Initialization option. Universal Server displays a series of six screens. Each screen contains a number of fields that correspond to parameters in the ONCONFIG configuration file.

Initializing Disk Space with oninit

After you configure Universal Server (refer to [Chapter 3, “Installing and Configuring”](#)), you can initialize disk space by executing one of the following commands:

```
% oninit -i
```

or

```
% oninit -i -s
```

The **oninit -i** option leaves Universal Server in on-line mode after initiation. If you use both the **-i** and **-s** options, Universal Server is left in quiescent mode. For reference information on executing **oninit**, see [“Initialize Disk Space and Shared Memory” on page 39-29](#).

Creating a Dbspace

This section explains how to create a standard dbspace (see [“What Is a Dbspace?” on page 14-16](#)) and a temporary dbspace (see [“What Is a Temporary Dbspace?” on page 14-21](#)). You can use ON-Monitor or **onspaces** for either task.

Important: *ON-Monitor supports operations only on dbspaces and blobspaces. You must use **onspaces** to manage sbspaces and extspaces.*

You can create a dbspace within ON-Monitor or from the command line. But you must first allocate disk space as described in [“Allocating Disk Space” on page 15-4](#).

You can mirror the dbspace when you create it if mirroring is enabled for Universal Server. Mirroring takes effect immediately.

Specify an explicit pathname for the initial chunk of the dbspace. Informix recommends that you use a linked pathname. (See [“Creating Links to Each Raw Device” on page 15-7](#).) If you are allocating a raw disk device, you might need to specify an offset to preserve track 0 information used by your UNIX operating system. (See [“Do You Need to Specify an Offset?” on page 15-6](#).) If you are allocating cooked disk space, the pathname is a file in a UNIX file system.



When the initial chunk of the dbspace that you are creating is cooked file space, Universal 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 Universal 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.

If you are creating a temporary dbspace, you must make Universal Server aware of the existence of the newly created temporary dbspace by setting the **DBSPACETEMP** configuration variable, the **DBSPACETEMP** environment variable, or both. Universal 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 Universal Server

You must be logged in as user **informix** or **root** to create a dbspace within ON-Monitor or from the command line.

If you are creating a standard (nontemporary) dbspace, Universal Server can be in on-line mode. The newly added dbspace (and its associated mirror, if one exists) is available immediately. If you are creating a temporary dbspace, you must reinitialize shared memory in order for Universal Server to write to that dbspace.

Creating a Dbspace with ON-Monitor

To create a dbspace with ON-Monitor, follow these instructions:

1. Select the Dbspaces menu, Create option to create a dbspace.
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.



Important: If you are creating a temporary dbspace, you must update the `DBSPACETEMP` configuration variable, the **`DBSPACETEMP`** environment variable, or both, and reinitialize shared memory; otherwise, Universal Server does not write to this dbspace.

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**. 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
```

For reference information on creating a temporary dbspace with **onspaces**, see [“Create a Blobspace, Dbspace, Extspace, or Temporary Dbspace” on page 39-56](#).

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, Extspace, or Temporary Dbspace” on page 39-56](#).



Important: If you are creating a temporary dbspace, you must update the `DBSPACETEMP` configuration variable, the **`DBSPACETEMP`** environment variable, or both, and reinitialize shared memory; otherwise, Universal Server does not write to this dbspace.

Adding a Chunk to a Dbspace

If one of your dbspaces is becoming full, you might want to add a new chunk. You can add a chunk to a dbspace within ON-Monitor or from the command line. Before you do, however, you must first allocate disk space as described in [“Allocating Disk Space” on page 15-4](#).

Adding a Chunk

You add a *chunk* when you need to increase the amount of disk space allocated to a dbspace, blobspace, or sbspace.

If you are adding a chunk to a mirrored dbspace, blobspace or sbspace, you must also add a mirrored chunk.

You must specify an explicit pathname for the chunk. Informix recommends that you use a linked pathname. See [“Creating Links to Each Raw Device” on page 15-7](#) for instructions.

When you add a chunk allocated as cooked file space, Universal 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, Universal 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 within ON-Monitor or from the command line.

You can make this change while Universal Server is in on-line mode. The newly added chunk (and its associated mirror, if one exists) is available immediately.

Adding a Chunk with ON-Monitor

To add a chunk to a dbospace, follow these instructions:

1. Select the Dbspaces menu, Add_chunk option.
2. Use RETURN or the Arrow keys to select the blobospace or dbospace that will receive the new chunk and press CTRL-B or F3.
3. The next screen indicates whether the blobospace or dbospace is mirrored. If it is, enter Y in the **Mirror** field.
4. If the dbospace to which you are adding the chunk is a temporary dbospace, enter Y in the **Temp** field.
5. If you indicated that the dbospace or blobospace 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.

Adding a Chunk Using onspaces

To add a chunk to a dbospace, use the **-a** option of **onspaces** as illustrated in the following example. This example adds a 10-megabyte mirrored chunk to **blobosp3**. 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 blobosp3 -p /dev/raw_dev1 -o 200 -s 10000 -m /dev/raw_dev2 200
```

For reference information on adding a chunk to a dbospace with **onspaces**, see [“Add a Chunk to a Blobospace or Dbospace” on page 39-64](#).

Creating a Blobspace

You can create a blobspace (see [“What Is a Blobspace?” on page 14-23](#)) with ON-Monitor or the **onspaces** utility. But you must first allocate disk space as described in [“Allocating Disk Space” on page 15-4](#).

Specify an explicit pathname for the initial chunk of the blobspace. Informix recommends that you use a linked pathname. For instructions on how to use a linked pathname, see [“Creating Links to Each Raw Device” on page 15-7](#).

You can mirror the blobspace when you create it if mirroring is enabled for Universal Server. Mirroring takes effect immediately.

A newly created blobspace is not immediately available for simple large object storage. Blobspace logging and recovery require that the statement that creates a blobspace and the statements that insert simple large objects 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 22-24](#).)

When the initial chunk of the blobspace that you are creating is cooked file space, Universal 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 cooked file that Universal Server created for the initial chunk is not removed. Its size represents the space left on your file system before you attempted to create the blobspace. Remove this file to reclaim the space.

You must be logged in as user **root** or user **informix** to create a blobspace with ON-Monitor or the **onspaces** utility.

You can create a blobspace while Universal Server is in on-line mode.

Before you create a blobspace, take time to determine what blobpage size is optimal for your environment. For instructions, see [“Optimizing Blobpage Size” on page 15-21](#).

Determining Universal Server Page Size

When you specify blobpage size, you specify it in terms of Universal Server pages. You can use one of the following methods to determine Universal Server page size for your system:

- Select the Shared-Memory option of the ON-Monitor Parameters menu. ON-Monitor displays a list of shared-memory parameters. Universal 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.
- To view the contents of the PAGE_PZERO reserved page, run the **oncheck -pr** utility.

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 Universal Server Page Size” on this page) per blobpage in the **BLOBPage Size** field. For example, if your Universal 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.

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, **blobbsp3**, with a blobpage size of 10 kilobytes, where Universal Server page size is 2 kilobytes. An offset of 200 kilobytes for the primary and mirrored chunks is specified.

```
% onspaces -c -b blobbsp3 -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, Dbospace, Extospace, or Temporary Dbospace”](#) on page 39-56.

Adding a Chunk to a Blobspace

Adding a chunk to a blobspace is identical to adding a chunk to a dbospace. Both procedures are explained in [“Adding a Chunk”](#) on page 15-12.

Creating a Sbspace

The procedure and considerations for creating a sbspace are similar to those for creating a blobspace. However, you can specify additional optional parameters with the **onspaces -Ms, -Mo, and -Df** options. The **-Ms** and **-Mo** options control the sizing and placement of metadata for an sbspace, while the **-Df** option lets you specify various default properties for smart large objects that will be stored in the sbspace.

To create a sbspace with **onspaces**, use the **-c** and **-S** option as illustrated in the following example:

```
% onspaces -c -S sblobsp4 -g 5 -p /dev/raw_dev1 -o 500 -s 20000 -m /dev/raw_dev2  
500 -Ms 150 -Mo 200 -Df AVG_LO_SIZE=32
```

This example creates a 20-megabyte mirrored sbspace, **sblobsp4**, with a sbpage size of 10 kilobytes, where Universal Server page size is 2 kilobytes. Offsets of 200 kilobytes for the primary and 500 kilobytes for the mirrored chunks are specified, as well as a metadata size of 150 kilobytes with a 200 kilobyte offset. The average expected smart-large-object size is 32 kilobytes.

For reference information on creating a sbspace with **onspaces**, see [“Create an Sbspace” on page 39-59](#). For information on sizing sbspace metadata, see “Sizing Sbspace Metadata” in the following section. For reference information on default options for smart large objects stored in a sbspace, see [“Creating a Sbspace with the -Df option” on page 39-61](#).

Sizing Sbspace Metadata

The metadata portion of an sbspace grows as smart large objects are added to that sbspace; therefore, it is important to size correctly the metadata area from a sbspace.

The easiest way to size the metadata area of a sbspace is to create the first sbspace chunk, specifying only the expected average size for the smart large objects that will be stored in the sbspace using the **onspaces -Df** option `AVG_LO_SIZE`. Universal Server automatically estimates the size of the metadata area for the first chunk and all subsequent chunks by dividing the estimated average size of the smart large objects by the size of the user data area. The first chunk in a sbspace must have a metadata area whether calculated by Universal Server or the user. Metadata is not allocated when you add an additional chunk to an existing sbspace.

The formula that the system uses to estimate the size of the metadata area in the preceding example presumes that the metadata area being allocated contains only the smart-large-object header partition for a single chunk, that is, the current chunk. The smart-large-object header partition contains one page (system page size, not sbpage size) for each smart large object in a chunk.

If some sbspace chunks are allocated without a metadata partition, using the **onspaces -a -U** option, a metadata area might contain the smart-large-object header partition for multiple sbspace chunks. In this case, the DBA can state the size of the metadata area explicitly to ensure that the sbspace does not run out of metadata space. When Universal Server exhausts the metadata area, it attempts to steal some space from the user data area.

A formula for sizing the metadata area is $\text{pagesize} * (\text{number of smart large objects} + 50 + (\text{number of chunks} * 28)) / 1024$. Because the metadata area is specified in kilobytes, you must multiply the metadata page count estimate by the system page size (typically 2 or 4 kilobytes).

For more information on the `onspaces` utility, see [“onspaces: Manage Blobspaces, Dbspaces, Extspaces, or Sbspaces”](#) on page 39-55.

Adding a Chunk to a Sbspace

Adding a chunk to a sbspace is identical to adding a chunk to a dbspace. However, two additional options are available. You can specify the size and offset of the sbspace metadata using the **-Ms** and **-Mo** options, or you can specify that the entire sbspace be used for user data using the **-U** option. This example adds a 10-megabyte mirrored chunk to `sblobsp4`. 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. The **-U** option specifies that the new chunk will contain user data exclusively

```
% onspaces -a sblobsp4 -p /dev/raw_dev1 -o 200 -s 10000 -m /dev/raw_dev2 200 -U
```

For reference information on adding a chunk to a dbspace with **onspaces**, see [“Add a Chunk to a Blobspace or Dbpace”](#) on page 39-64. For more information on sizing sbspace metadata, see [“Sizing Sbspace Metadata”](#) on page 15-17.

Dropping a Chunk from a Dbpace with onspaces

To drop a chunk successfully from a dbpace 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 Universal Server entity still occupies space in the chunk, remove it, and reenter the **onspaces** command.

You cannot drop the initial chunk of a dbpace. Use the `fchunk` column of **onstat -d** to determine which chunk is the initial chunk of a dbpace. For more information, see [“-d Option”](#) on page 39-84.

The following example drops a chunk from **dbsp3**. 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 39-67](#).

Dropping a Chunk from a Blobspace or Sbspace

The procedure for dropping a chunk from a blobspace or sbspace is identical to the procedure for dropping a chunk from a dbspace described in [“Dropping a Chunk from a Dbpace with onspaces” on page 15-18](#) except that Universal Server must be in quiescent mode. Other than this condition, you need only substitute the name of your blobspace or sbspace wherever a reference to a dbpace occurs.

Dropping a Dbpace, Blobspace, or Sbspace

Before you drop a dbpace, you must first drop all databases and tables that you previously created in the dbpace. Before you drop a blobspace, you must drop all tables that have a TEXT or BYTE column that references the blobspace. Likewise, before you can drop a sbspace, you must drop all tables that have a CLOB or BLOB column that references objects that are stored in the sbspace.

For sbspaces, you need not delete columns that *point* to a sbspace, but these columns must be null; that is, all smart large objects must be deallocated from the sbspace.

Execute **oncheck -pe** to verify that no tables or log files are residing in the dbpace or blobpace.

You cannot drop the root dbpace.



Warning: After you drop a dbspace, blobspace, or sbspace, the newly freed chunks are available for reassignment to other dbspaces, blobspaces, or sbspaces. However, before you reassign the newly freed chunks, you must perform a 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 the [“INFORMIX-Universal Server Backup and Restore Guide.”](#) If you are using ON-Archive or the **ontape** utility as your backup and restore tool, refer to the [“INFORMIX-Universal Server Archive and Backup Guide.”](#)

If you drop a dbspace, blobspace, or sbspace that is mirrored, the dbspace, blobspace, or sbspace mirrors are also dropped.

If you want to drop only the dbspace, blobspace, or sbspace mirrors, turn off mirroring. (See [“Ending Mirroring” on page 28-12.](#)) This action drops the dbspace, blobspace, or sbspace 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 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.

Dropping a Dbspace, Blobspace, or Sbspace with onspaces

To drop a dbspace, blobspace, or sbspace 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
```

This example drops a sbpace called **sbspc4** and its mirrors.

```
% onspaces -d sbspc4
```

For reference information on dropping a dbspace, blobspace, or sbpace with **onspaces**, see [“Drop a Blobspace, Dbspace, Extspace, or Sbspace” on page 39-63](#).

Optimizing Blobspace Blobpage Size

Familiarize yourself with the Universal Server approach to blobspace simple large object storage before you begin this section. [“Structure of a Blobspace” on page 42-62](#) and [“Blobspace Page Types” on page 42-67](#) provide background information for this section. This section is not applicable if you store simple large objects 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 simple large object

Blobpage fullness refers to the amount of data within each blobpage. simple large objects stored in a blobspace cannot share blobpages. Therefore, if a single simple large 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 simple large object, you can 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 Universal 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 simple large object stored as part of the table (or database)

The **oncheck -pe** command can provide background information about the simple large 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 Universal Server pages used by each table to store its associated simple large object 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 simple large objects in a database or table. If you find that the statistics for a significant number of simple large objects show a low percentage of fullness, Universal Server might benefit from resizing the blobpage in the blobspace.

The following example retrieves storage information for all simple large objects stored in the table **sriram.catalog** in the **stores7** database:

```
% oncheck -pB stores7:sriram.catalog
```

Figure 15-2 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 15-2
Output of
oncheck -pB

Space Name is the name of the blobspace that contains one or more simple large objects stored as part of the table (or database).

Page Number is the starting address in the blobspace of a specific simple large object.

Pages is the number of Universal Server pages required to store this simple large object.

Percent Full is a measure of the average fullness of all the blobpages that hold this simple large object.

Page Size is the size in bytes of the blobpage for this blobspace. Blobpage size is always a multiple of the Universal Server page size. See “[Determining Universal Server Page Size](#)” on page 15-15 for instructions on how to obtain the page size for your database server.

The example output indicates that four simple large objects are stored as part of the table **sriram.catalog**. Two simple large objects are stored in the blobspace **blobPIC** in 6144-byte blobpages. Two more simple large 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 simple large objects. The total says nothing about the size of the blobpages used, the number of simple large 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 simple large object storage.

The following sections use the output shown in [Figure 15-2 on page 15-23](#) to demonstrate the idea of average fullness.

Interpreting Blobpage Average Fullness

The first simple large object listed in [Figure 15-2 on page 15-23](#) is stored in the blobspace **blobPIC** and requires one 6144-byte blobpage. The blobpage is 51 to 75 percent full, meaning that the simple large object size is between $0.51 * 6144 = 3133$ bytes and $0.75 * 6144 = 4608$. The maximum size of this simple large object must be less than or equal to 75 percent of 6144 bytes, or 4608 bytes.

The second simple large 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 simple large object must be greater than 50 percent of 12,288 bytes, or 6144 bytes. The maximum size of the simple large 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 simple large objects in blobspace **bspc1**. These two simple large objects appear to be nearly the same size. Both simple large objects require two 2048-byte blobpages, and the average fullness for each is 76 to 100 percent. The minimum size for these simple large objects must be greater than 75 percent of the allocated blobpages, or 3072 bytes. The maximum size for each simple large object is slightly less than 4096 bytes (allowing for overhead).

Apply Efficiency Criteria to Output

Looking at the efficiency information for blob space **bspc1**, a Universal Server administrator might decide that a better simple-large-object storage strategy would be to double the blobpage size from 2048 bytes to 4096 bytes. (Blobpage size is always a multiple of the Universal Server page size.) If the Universal Server administrator made this change, the measure of page fullness would remain the same, but the number of locks needed during a simple large object update or modification would be reduced by half.

The efficiency information for blob space **blobPIC** reveals no obvious suggestion for improvement. The two simple large objects in **blobPIC** differ considerably in size, and there is no optimal storage strategy. In general, simple large objects of similar size can be stored more efficiently than simple large objects of different sizes.

Managing Extspaces

An extspace (see [“What Is a Extspace?” on page 14-25](#)) does not require allocation of disk space. You create and drop extspaces using the **onspaces** utility.

Creating a Extspace

You create a extspace with the **onspaces** utility. But you should first have a valid datasource and a valid access method with which to access that data source. Although you can create an extspace without a valid access method or a valid datasource, any attempts to retrieve data from the extspace will generate an error. For information on access methods, see the *Virtual-Table Interface Programmer’s Manual*.

To create a extspace with **onspaces**, use the **-c** option as illustrated in the following example. This example creates a extspace, `pass_space`, that is associated with the UNIX password file.

```
% onspaces -c -x pass_space -l /etc/passwd
```



Important: The preceding example assumes that you have coded a routine that provides functions for correctly accessing the file **passwd** and that the file itself exists. Once you have created the extspace, you must use the appropriate commands to allow access to the data in the file **passwd**. For more information on user-defined access methods, see the “Virtual-Table Interface Programmer’s Manual.”

For reference information on creating a extspace with **onspaces**, see [“Create a Blobspace, Dbspace, Extspace, or Temporary Dbspace”](#) on page 39-56.

Dropping a Extspace

To drop a extspace with **onspaces**, use the **-d** option as illustrated in the following examples. A extspace cannot be dropped if it is associated with an existing table or index.

This example drops a extspace called **pass_space**.

```
% onspaces -d pass_space
```

What Is Fragmentation?

What Is Fragmentation?	16-3
Why Use Fragmentation?	16-5
Whose Responsibility Is Fragmentation?	16-6
Fragmentation and Logging	16-7
Can You Fragment a Temporary Table?	16-7
Distribution Schemes for Table Fragmentation	16-7
Round-Robin Distribution Scheme	16-9
Round-Robin Fragmentation of Smart Large Objects	16-9
Expression-Based Distribution Schemes	16-10
Types of Expression-Based Distribution Schemes.	16-10
When Can the Database Server Eliminate Fragments from a Search?	16-12
Expression-Based Distribution Schemes for Fragment Elimination	16-12
Effectiveness of Fragment Elimination	16-16
Fragmentation of Table Indexes	16-17
Attached Index.	16-17
Detached Index.	16-18
Restrictions on Indexes for Fragmented Tables	16-18
Internal Structure of Fragmented Tables	16-19
Disk Allocation.	16-19
Rowids	16-20
Creating a Rowid Column.	16-21
What Happens When You Create a Rowid Column?	16-21
Skipping Inaccessible Fragments	16-21
Effect of the Data-Skip Feature on Transactions.	16-22
When to Use Data Skip	16-22
When to Skip Selected Fragments	16-23
When to Skip All Fragments	16-23

Formulating a Fragmentation Strategy	16-23
Deciding How To Fragment a Table	16-25
Fragmenting for Single-User Response Time	16-25
Fragmenting for Improved Concurrency	16-26
Fragmenting for Improved Availability of Data	16-27
Fragmenting for Improved Backup-and-Restore Characteristics	16-28
Which Distribution Scheme Should You Use?	16-29
Choosing Round-Robin or Expression-Based Distribution Schemes	16-30
Designing an Expression-Based Distribution Scheme	16-31
How Many Fragments Should You Use?	16-32

F

ragmentation allows you to control where data is stored at the table level. The logical and physical units that INFORMIX-Universal Server uses for data storage are discussed in [Chapter 14, “Where Is Data Stored?”](#) Fragmentation is an extension of that topic.

The following fragmentation topics are covered in this chapter:

- Overview of the fragmentation feature
- Distribution schemes for fragmentation
- Fragmentation of table indexes
- Skipping inaccessible fragments during a query
- Formulating a fragmentation strategy

What Is Fragmentation?

Fragmentation is a Universal 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 associated with a specific physical disk. You create the fragments and assign them to dbspaces with SQL statements.

The scheme that you use to group rows or index keys into fragments is called the *distribution scheme*. The distribution scheme and the set of dbspaces in which you locate the fragments together make up the *fragmentation strategy*. The decisions that you must make to formulate a fragmentation strategy are discussed in [“Formulating a Fragmentation Strategy” on page 16-23](#).

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 contained in fragmented tables.

What Is Fragmentation?

Universal 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 contained in this system catalog table, refer to the [Informix Guide to SQL: Reference](#).

Because Universal Server has information on which fragments contain which data, Universal Server can route client requests for data to the appropriate fragment without accessing irrelevant fragments as illustrated in Figure 16-1. (This statement is not true for the round-robin distribution scheme or all expression-distribution schemes, however. For more information, see “[Distribution Schemes for Table Fragmentation](#)” on page 16-7.)

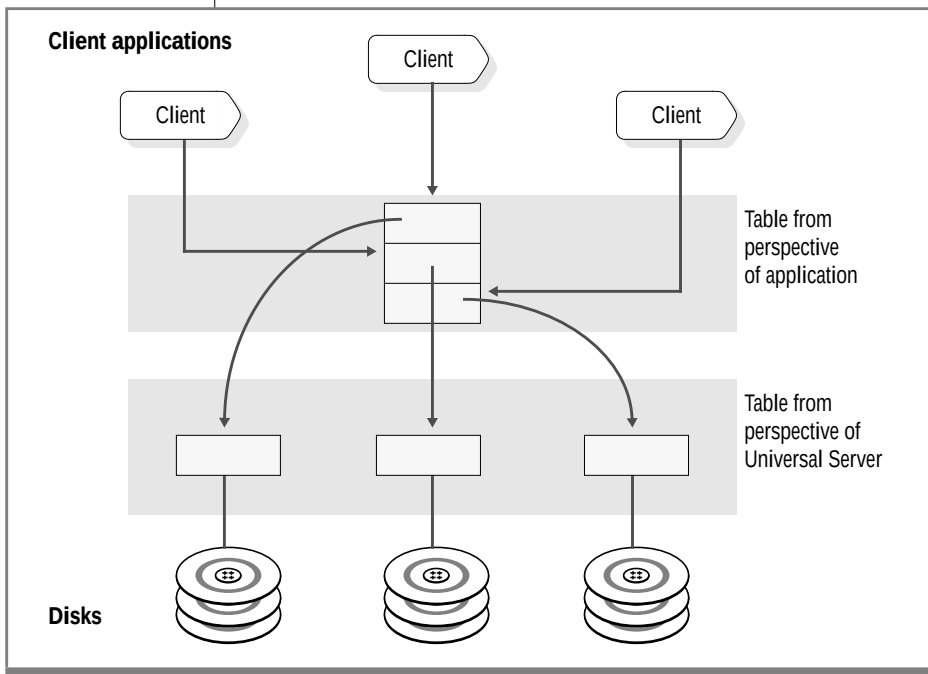


Figure 16-1
*Universal Server
Routes Requests for
Data from Client
Applications*

Why Use Fragmentation?

Consider fragmenting your tables if you have at least one of the following goals:

- Improved single-user response time

You can improve the performance of individual queries by using fragmentation with parallel database query (PDQ) to scan fragments spread across multiple disks in parallel. See [“What Is PDQ?” on page 18-4](#).

When Universal Server uses PDQ to process a query, it allocates threads to retrieve data from fragments in parallel. You can expect to see a near-linear improvement in performance with each fragment (located on a separate device) that you add to a table. For more information, see [“Fragmenting for Single-User Response Time” on page 16-25](#).

- Improved concurrency

Fragmentation can reduce contention for data located in large, high-use tables. Fragmentation reduces contention because each fragment resides on a separate I/O device, and Universal Server directs queries to the appropriate fragment. For more information, see [“Fragmenting for Improved Concurrency” on page 16-26](#).

- Improved availability

You can improve the availability of your data by using fragmentation. If a fragment becomes unavailable, Universal Server is still able to access the remaining fragments. See [“Skipping Inaccessible Fragments” on page 16-21](#).

- Improved backup-and-restore characteristics

Fragmentation gives you a finer backup-and-restore granularity. This granularity can reduce the time 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, by having these operations performed in parallel. For more information, see [“Fragmenting for Improved Backup-and-Restore Characteristics” on page 16-28](#). For more information about the ON-Bar backup and restore system, refer to the *INFORMIX-Universal Server Backup and Restore Guide*. For more information on ON-Archive, refer to the *INFORMIX-Universal Server Archive and Backup Guide*.

Each of the preceding goals has its own implications for the fragmentation strategy that you ultimately implement. These issues are discussed under [“Formulating a Fragmentation Strategy” on page 16-23](#). Your primary fragmentation goal will determine, or at least influence, 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.

Whose Responsibility Is Fragmentation?

Some overlap exists between the responsibilities of the Universal 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 Universal Server administrator, on the other hand, has the responsibility of laying out the disk space in which the fragmented tables will reside. Because neither of these responsibilities can be performed in isolation of the other, implementing fragmentation requires a cooperative effort between the Universal Server administrator and the DBA.

Fragmentation and Logging

A fragmented table can belong to either a database that uses logging or to a database that does not use logging. As with nonfragmented tables, if a fragmented table is part of a nonlogging database, a potential for data inconsistencies arises if a failure occurs.

Can You Fragment a Temporary Table?

You can create a temporary, fragmented table with the `TEMP TABLE` clause of the `CREATE TABLE` statement. Universal Server deletes the fragments created for a temp table at the same time that it deletes the temporary table.

One restriction for fragmenting temporary tables is that you cannot alter the fragmentation strategy of a temporary table (as you can with permanent tables).

Distribution Schemes for Table Fragmentation

A distribution scheme is a method that Universal Server uses to distribute rows or index entries to fragments.

Universal Server supports the following two types of distribution schemes:

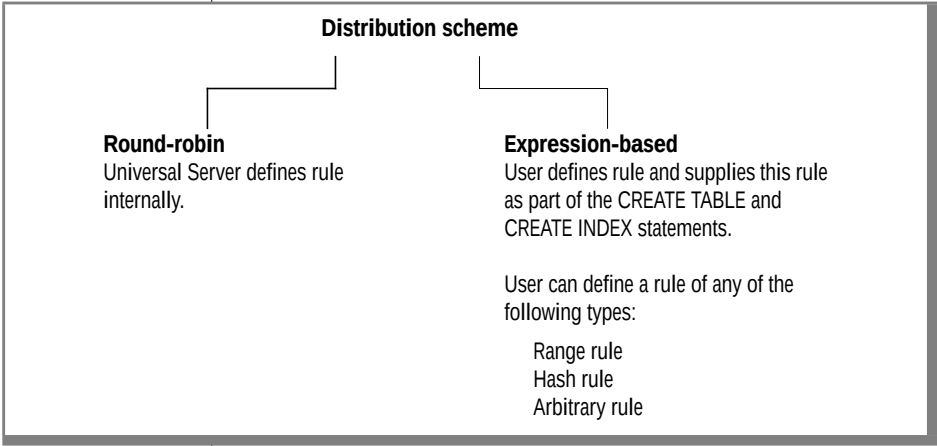
- Round-robin distribution scheme
- Expression-based distribution scheme

The round-robin distribution scheme uses a rule (algorithm) defined internally by Universal Server to place rows or index keys in fragments. An expression-based distribution scheme requires the user to define the rule and include it in the `CREATE TABLE` or `CREATE INDEX` statement, as shown in [Figure 16-2 on page 16-8](#).

For an expression-based distribution scheme, the user can provide a rule definition that belongs to one of the following three classes:

- Range rule
- User-defined hash rule
- Arbitrary rule

Figure 16-2
*Possible
Distribution
Schemes and
Fragmentation
Rules*



The following sections describe the round-robin distribution scheme and the different types of expression-based distribution schemes. For complete descriptions of the SQL syntax of these distribution schemes, refer to the CREATE TABLE and CREATE INDEX statements in the [Informix Guide to SQL: Syntax](#).

Round-Robin Distribution Scheme

To use a round-robin distribution scheme, use the `FRAGMENT BY ROUND ROBIN` clause of the `CREATE TABLE` statement, as follows:

```
CREATE TABLE account...FRAGMENT BY ROUND ROBIN IN dbspace1, dbspace2, dbspace3
```

When Universal Server receives a request to insert a number of rows into a table that is fragmented by round-robin, it distributes the rows in such a way that the number of rows in each of the fragments remains approximately the same. Round-robin distributions are also called *even distributions* because information is distributed evenly among the fragments. The rule for distributing rows to tables fragmented by round-robin is internal to Universal Server.



Important: You can use the round-robin distribution scheme only to fragment table rows. You cannot fragment a table index with this distribution scheme. See [“Fragmentation of Table Indexes” on page 16-17](#).

Round-Robin Fragmentation of Smart Large Objects

You can specify multiple sbspaces in the `PUT` clause of the `CREATE TABLE` statement to achieve round-robin fragmentation of smart large objects on a column basis. If you specify multiple sbspaces for a CLOB or BLOB column, Universal Server will distribute the smart large objects for the column to the specified sbspaces in round-robin fashion. The following `CREATE TABLE` statement, for example, causes Universal Server to distribute **cat_picture** to **sbcatpic1**, **sbcatpic2**, and **sbcatpic3** in round-robin fashion.

```
CREATE TABLE catalog (
    catalog_num SERIAL,
    stock_num    SMALLINT,
    manu_code    CHAR(3),
    cat_descr    VARCHAR(255, 65),
    cat_picture  BLOB)
PUT cat_picture in (sbcatpic1, sbcatpic2, sbcatpic3)
```

Expression-Based Distribution Schemes

To use an expression-based distribution scheme, use the `FRAGMENT BY EXPRESSION` clause of the `CREATE TABLE` or `CREATE INDEX` statement. This clause takes the following form:

```
CREATE TABLE tablename ... FRAGMENT BY EXPRESSION
    <expression_1> in dbspace_1
    <expression_2> in dbspace_2
    .
    .
    .
    <expression_n> in dbspace_n;
```

When you use the `FRAGMENT BY EXPRESSION` clause of the `CREATE TABLE` statement to create a fragmented table, you must supply one condition for each fragment of the table that you are creating.

You can define *range rules*, *hash rules*, or *arbitrary rules* that indicate to Universal Server how rows are to be distributed to fragments. All three types of expression-based distribution schemes are explained with examples under “Types of Expression-Based Distribution Schemes” in the following section.

Types of Expression-Based Distribution Schemes

This section presents an overview of the three types of expression-based distribution schemes.

Range Rule

A range rule uses SQL relational and logical operators to define the boundaries of each fragment in a table. A range rule can contain the following restricted set of operators:

- The relational operators `>`, `<`, `>=`, `<=`
- The logical operator `AND`

A range rule can refer to only one column in a table but can make multiple references to this column. You define one expression for each of the fragments in your table, as shown in the following example:

```
.
.
.
FRAGMENT BY EXPRESSION
id_num > 0 AND id_num <= 20 IN dbsp1,
id_num > 20 AND id_num <= 40 IN dbsp2,
id_num > 40 IN dbsp3
```

User-Defined Hash Rule

A hash rule maps each row in a table to a set of integers called hash values. Universal Server uses these values to determine in which fragment it will store a given row. User-defined hash rules are usually implemented with the MOD function, as shown in the following example:

```
.
.
.
FRAGMENT BY EXPRESSION
MOD(id_num, 3) = 0 IN dbsp1,
MOD(id_num, 3) = 1 IN dbsp2,
MOD(id_num, 3) = 2 IN dbsp3
```

Arbitrary Rule

An arbitrary rule uses SQL relational and logical operators. Unlike range rules, arbitrary rules allow you to use any relational operator and any logical operator to define the rule. In addition, you can reference any number of table columns in the rule. Arbitrary rules typically include the use of the OR logical operator to group data, as shown in the following example:

```
.
.
.
FRAGMENT BY EXPRESSION
zip_num = 95228 OR zip_num = 95443 IN dbsp2,
zip_num = 91120 OR zip_num = 92310 IN dbsp4,
REMAINDER IN dbsp5
```

Inserting and Updating Rows in Expression-Based Fragments

When you insert or update a row, Universal Server evaluates fragment expressions, in the order specified, to see if the row belongs in any of the fragments. If so, Universal Server inserts or updates the row in one of the fragments. If the row does not belong in any of the fragments, the row is put into the fragment specified by the remainder clause. If the distribution scheme does not include a remainder clause, and the row does not match the criteria for any of the existing fragment expressions, an error is returned.

When Can the Database Server Eliminate Fragments from a Search?

If you use an appropriate distribution scheme, Universal Server can eliminate fragments from a search before it performs the actual search. This capability can improve performance significantly and reduce contention for the disks on which fragments reside.

Whether Universal Server can eliminate fragments from a search depends on the fragmentation expression of the table that is being searched *and* the form of the query expression (the expression in the WHERE clause).

Two types of query expressions are considered for fragment elimination: range expressions and equality expressions. Consider the following query-expression template:

```
column operator value
```

Range expressions use a relational operator (<, >, <=, >=), and equality expressions use an equality operator (=, IN). The *value* in either type of query expression must be a literal, a host variable, a stored procedure variable, or a noncorrelated subquery.

Expression-Based Distribution Schemes for Fragment Elimination

Universal Server cannot eliminate fragments when you fragment a table with a round-robin distribution scheme. Furthermore, not all expression-based distribution schemes give you the same fragment-elimination behavior. The following sections discuss ways of fragmenting data to improve fragment elimination behavior.

Nonoverlapping Fragments on a Single Column

A fragmentation rule that creates nonoverlapping fragments on a single column is the preferred fragmentation rule from a fragment-elimination standpoint. The advantage of this type of distribution scheme is that Universal Server can eliminate fragments for queries with range expressions as well as queries with equality expressions. Informix recommends that you meet these conditions when you design your fragmentation rule. Figure 16-3 gives an example of this type of fragmentation rule.

```
.  
. .  
. .  
  
FRAGMENT BY EXPRESSION  
a <= 8 OR a IN (9,10) IN dbsp1,  
10 < a <= 20 IN dbsp2,  
a IN (21,22, 23) IN dbsp3,  
a > 23 IN dbsp4;
```

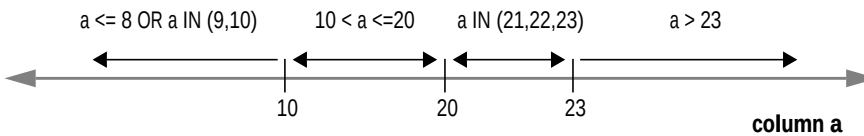


Figure 16-3
*Schematic Example
of Nonoverlapping
Fragments on a
Single Column*

You can create nonoverlapping fragments using a range rule or an arbitrary rule based on a single column. You can use relational operators, as well as AND, IN, OR, or BETWEEN. Be careful when you use the BETWEEN operator because when Universal Server parses the BETWEEN keyword, it includes the endpoints you specify in the range of values. Avoid using a REMAINDER clause in your expression. If you use a REMAINDER clause, Universal Server is not always able to eliminate the remainder fragment.

Overlapping Fragments on a Single Column

The only restriction for this category of fragmentation rule is that you base the fragmentation rule on a single column. The fragments can be overlapping and noncontiguous. You can use any range, hash, or arbitrary rule that is based on a single column. Figure 16-4 gives an example of this type of fragmentation rule.

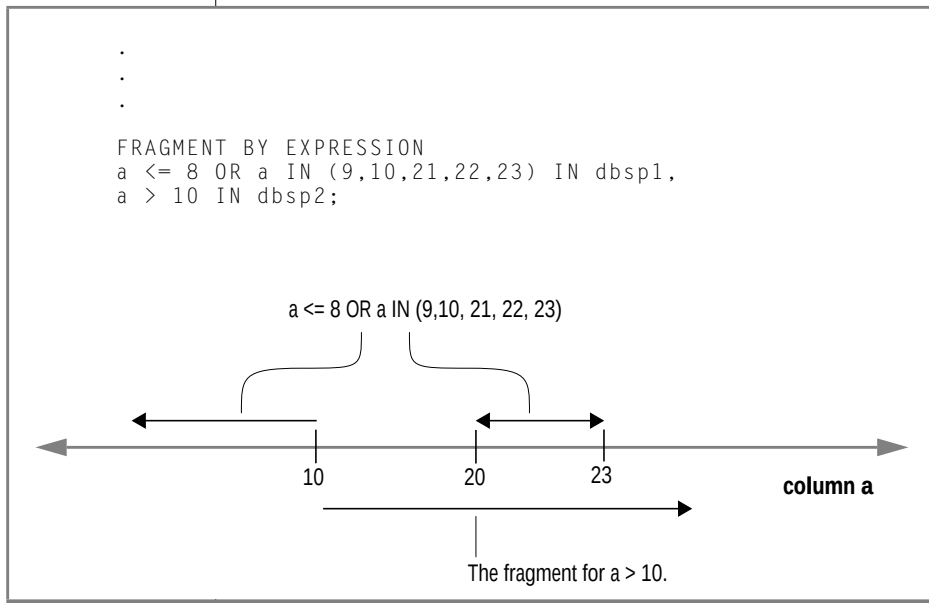


Figure 16-4
*Schematic Example
of Overlapping
Fragments on a
Single Column*

If you use this type of distribution scheme, Universal Server can eliminate fragments on an equality search but not a range search. This can still be very useful because all INSERT and many UPDATE operations perform equality searches.

This alternative is acceptable if you cannot use an expression that creates nonoverlapping fragments with contiguous values. For example, in cases where a table is growing over time, you might want to use a hash rule to keep the fragments of similar size. Expression-based distribution schemes that use hash rules fall into this category because the values in each fragment are not contiguous.

Nonoverlapping Fragments, Multiple Columns

This category of expression-based distribution scheme uses an arbitrary rule to define nonoverlapping fragments based on multiple columns. Figure 16-5 gives an example of this type of fragmentation rule.

```
.  
. .  
. .  
. .  
  
FRAGMENT BY EXPRESSION  
0 < a <= 10 AND b IN ('E', 'F','G') IN dbsp1,  
0 < a <= 10 AND b IN ('H', 'I','J') IN dbsp2,  
10 < a <= 20 AND b IN ('E', 'F','G') IN dbsp3,  
10 < a <= 20 AND b IN ('H', 'I','J') IN dbsp4,  
20 < a <= 30 AND b IN ('E', 'F','G') IN dbsp5,  
20 < a <= 30 AND b IN ('H', 'I','J') IN dbsp6;
```

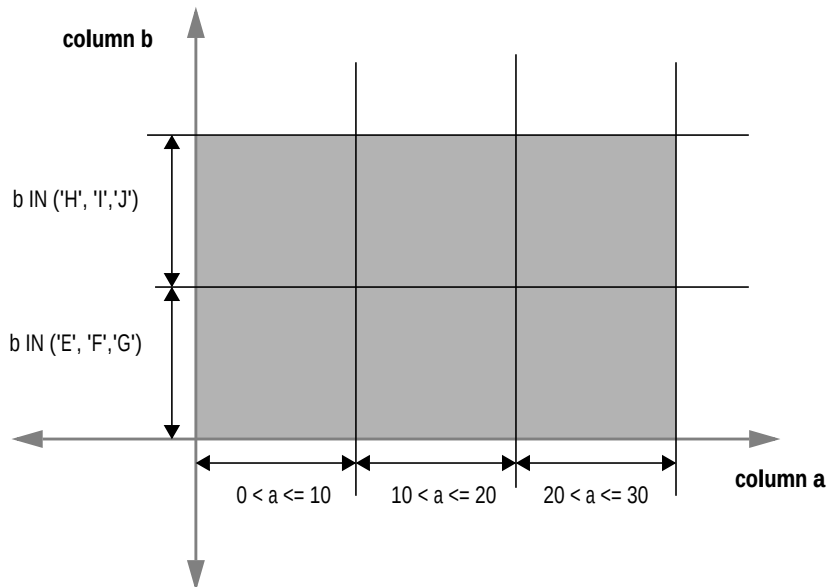


Figure 16-5
*Schematic Example
of Nonoverlapping
Fragments on Two
Columns*

If you use this type of distribution scheme, Universal Server can eliminate fragments on an equality search but not a range search. Again, this capability can still be very useful because all INSERT operations and many UPDATE operations perform equality searches. Avoid using a REMAINDER clause in the expression. If you use a REMAINDER clause, Universal Server is not always able to eliminate the remainder fragment.

This alternative is acceptable if you cannot obtain sufficient granularity using an expression based on a single column.

Effectiveness of Fragment Elimination

Figure 16-6 summarizes fragment-elimination behavior for different combinations of fragmentation scenarios and queries expressions.

Figure 16-6
Fragment Elimination

	Nonoverlapping Fragments on a Single Column	Overlapping or Noncontiguous Fragments on a Single Column	Nonoverlapping Fragments, Multiple Columns
Range Expression	Fragments can be eliminated.	Fragments <i>cannot</i> be eliminated.	Fragments <i>cannot</i> be eliminated.
Equality Expression	Fragments can be eliminated.	Fragments can be eliminated.	Fragments can be eliminated.

Importantly, where Figure 16-6 indicates that Universal Server can eliminate fragments, the effectiveness of fragment elimination is determined by the WHERE clause of the query in question. For example, consider a table fragmented with the following expression:

```
.  
.   
.   
FRAGMENT BY EXPRESSION  
100 < column_a AND column_b < 0 IN dbbsp1,  
100 >= column_a AND column_b < 0 IN dbbsp2,  
column_b >= 0 IN dbbsp3
```

Consider a WHERE clause that has the following expression:

```
column_a = 5 OR column_b = -50
```

Universal Server cannot eliminate any of the fragments from the search. On the other hand, consider a WHERE clause that has the following expression:

```
column_b = -50
```

Universal Server can eliminate the last fragment. Furthermore, consider a WHERE clause that has the following expression:

```
column_a = 5 AND column_b = -50
```

Universal Server can eliminate the last two fragments.

Fragmentation of Table Indexes

When you fragment a table, the indexes that are associated with that table are fragmented implicitly, according to the fragmentation scheme that you use for the table. You can also use the `FRAGMENT BY` clause of the `CREATE INDEX` statement to fragment the index for any table explicitly. The fragmentation scheme for an index can differ from that of the table. You can fragment the index for any table by the expression distribution scheme.

You can fragment both table data and table indexes. The fragmentation strategy for an index can be the same as the table-data fragmentation strategy (attached) or independent of the table-data fragmentation strategy (detached).

Attached Index

An index is *attached* when it follows the fragmentation strategy of the table data. If you create an index on a fragmented table but do not specify a distribution scheme for that index, Universal Server fragments the index according to the same distribution scheme as that table by default. That is, the index keys are distributed into fragments using the same rule as the table data and placed into the same dbspaces as the corresponding table data.

Detached Index

When you create an index, you can specify a fragmentation strategy for that index (a distribution scheme for the index keys and a set of dbspaces to contain the fragments). You have the same choice of expression-based distribution schemes that you do for table data (see the CREATE INDEX statement of the [Informix Guide to SQL: Syntax](#)); however, you cannot use the round-robin distribution scheme for an index.

You can change the fragmentation strategy of a fragmented index with the ALTER FRAGMENT ON INDEX statement.

If you do not want the index on a fragmented table to be fragmented, you can place the index in a separate dbspace with the CREATE INDEX... IN DBSPACE statement.

Restrictions on Indexes for Fragmented Tables

If Universal Server scans a fragmented index, multiple index fragments must be scanned and the results merged together. (The exception is if the index is fragmented according to some index-key range rule, and the scan does not cross a fragment boundary.) Because of this requirement, performance on index scans might suffer if the index is fragmented.

Because of these performance considerations, Universal Server places the following restrictions on indexes:

- You cannot fragment indexes by round-robin.
- You cannot fragment unique indexes by an expression that contains columns that are not in the index key.

For example, the following statement is illegal:

```
CREATE UNIQUE INDEX ia on tabl(col1)
  FRAGMENT BY EXPRESSION
    col2<10 in dbsp1,
    col2>=10 AND col2<100 in dbsp2,
    col2>100 in dbsp3;
```


Internal Structure of Fragmented Tables

Although table fragmentation is transparent to applications, as Universal Server administrator you should be aware of how Universal Server allocates disk space for table fragments and how Universal Server identifies rows in those fragments.

Disk Allocation

Each table fragment has its own tblspace with a unique *tblspace_id* or *fragment_id*. Figure 16-7 shows the disk allocation for a fragmented table.

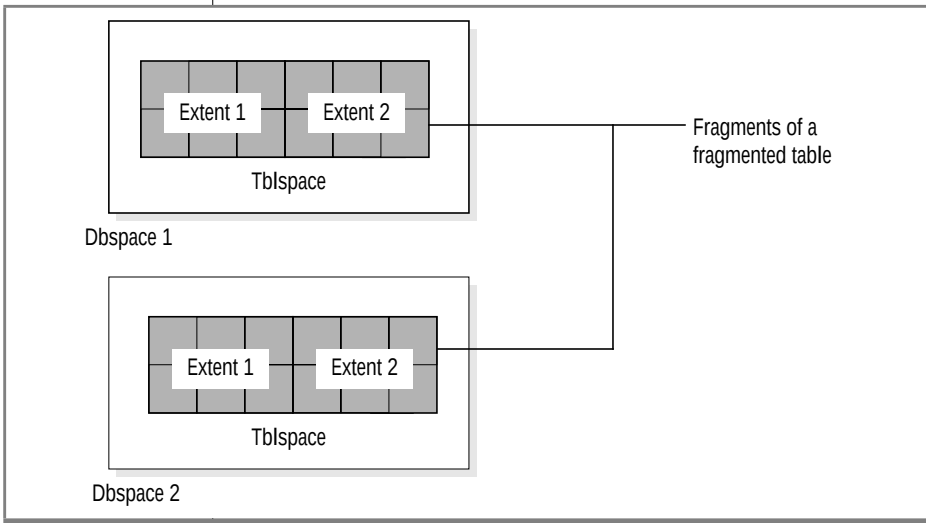
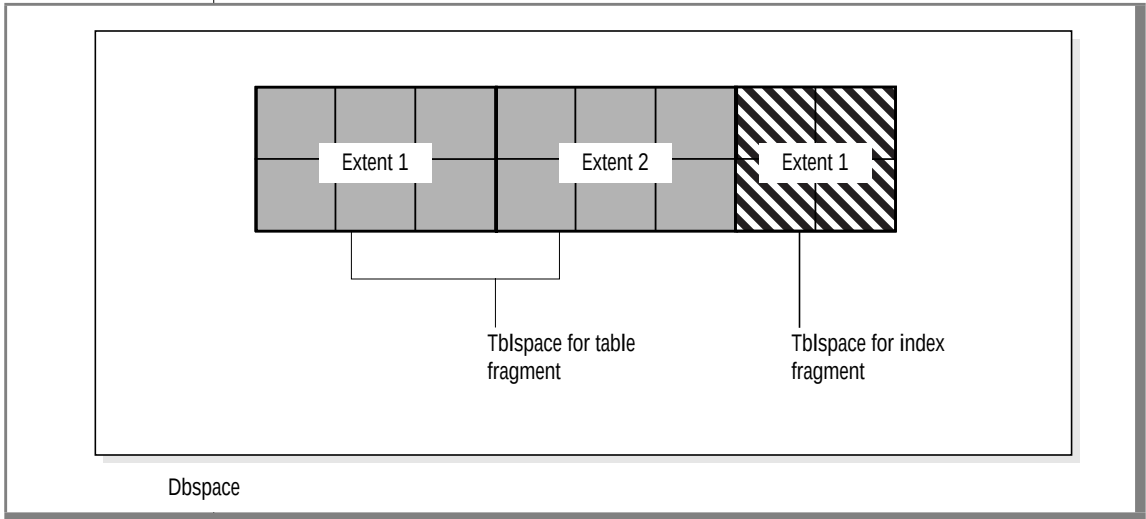


Figure 16-7
*Disk Structures for a
Fragmented Table*

If a fragmented table has an attached index, the data pages and index pages are not mixed within the same extent as they are with a nonfragmented table. In fact, the table fragment and attached index fragment are stored in separate tblspaces as shown in [Figure 16-8 on page 16-20](#).

Universal Server bases the extent size for the attached index tblspace on the extent size for the corresponding table fragment. Universal Server uses the ratio of the row size to index-key size (including the internal index-key overhead) to assign an appropriate extent size for the attached index tblspace.

Figure 16-8
Disk Allocation for a Table Fragment with an Attached Index Fragment



Rowids

The rowid of a row in a nonfragmented table is a unique and constant value that points to a location on disk. (See [“What Is a Rowid?”](#) on page 42-37.) This statement is not true for the rows in fragmented tables. The recommended method for referencing a row is to use the primary-key value. Primary keys are defined in the ANSI specification of SQL. Using primary keys makes applications more portable.

To accommodate applications that must reference a rowid for a fragmented table, Universal Server allows you to explicitly create a rowid column for a fragmented table. Accessing a row using an explicitly created rowid column is slower than using a primary key.

Creating a Rowid Column

To create the rowid column, use the following SQL syntax:

- The WITH ROWIDS clause of the CREATE TABLE statement
- The ADD ROWIDS clause of the ALTER TABLE statement
- The INIT clause of the ALTER FRAGMENT statement

You cannot create the rowid column by naming it as one of columns in a table you create or alter.

What Happens When You Create a Rowid Column?

When you create the rowid column, Universal Server takes the following actions:

- Adds the 4-byte unique value to each row in the table
- Creates an internal index that it uses to access the data in the table by rowid
- Inserts a row in the **sysfragments** catalog table for the internal index

Skipping Inaccessible Fragments

An additional benefit provided by fragmentation is that during an I/O operation, Universal Server can skip table fragments that are unavailable (for example, a fragment that is located on a chunk that is down). Therefore, a disk failure affects only a portion of the data in the fragmented table. In contrast, tables that are not fragmented can become completely inaccessible if they are located on a disk that fails.

This functionality is controlled as follows:

- By the Universal Server administrator with the DATASKIP configuration parameter
- By individual applications with the SET DATASKIP statement

For more information, see [“Skipping Fragments” on page 17-4](#).

Effect of the Data-Skip Feature on Transactions

If you turn the data-skip feature on, a SELECT statement always executes. In addition, an INSERT statement always succeeds if the table is fragmented by round-robin and at least one fragment is on-line. However, Universal Server *does not* complete operations that write to the database if a possibility exists that such operations might compromise the integrity of the database. The following operations fail:

- All UPDATE and DELETE operations where Universal Server cannot eliminate the down fragments
If Universal Server *can* eliminate the down fragments, the update or delete will be successful, but this outcome is independent of the DATASKIP setting.
- An INSERT operation for a table fragmented according to an expression-based distribution scheme where the appropriate fragment is down
- Any operation that involves referential constraint checking if the constraint involves data in a down fragment
For example, if an application deletes a row that has child rows, the child rows must also be available for deletion.
- Any operation that affects an index value (for example, updates to a column that is indexed) where the index in question is located in a down chunk

When to Use Data Skip

Use this feature sparingly and with caution because the results are always suspect. Consider using it in the following situations:

- You can accept the compromised integrity of transactions.
- You can determine that the integrity of the transaction will not be compromised.

The latter task can be difficult and time consuming.

When to Skip Selected Fragments

In certain circumstances, you might want Universal Server to skip over some fragments but not others. This usually occurs in the following situations:

- Fragments can be skipped because they do not contribute significantly to a query result.
- Certain fragments are down, and you decide that skipping these fragments and returning a limited amount of data is preferable to canceling a query.

When you want to skip fragments, use the ON *dbspace-list* setting to specify a list of dbspaces that contain the fragments that Universal Server should skip over.

When to Skip All Fragments

Setting the DATASKIP configuration parameter to ALL causes Universal Server to skip all unavailable fragments. Use this option with caution. If a dbspace becomes unavailable, all queries initiated by applications that do not issue a SET DATASKIP OFF statement before they execute could be subject to errors.

Formulating a Fragmentation Strategy

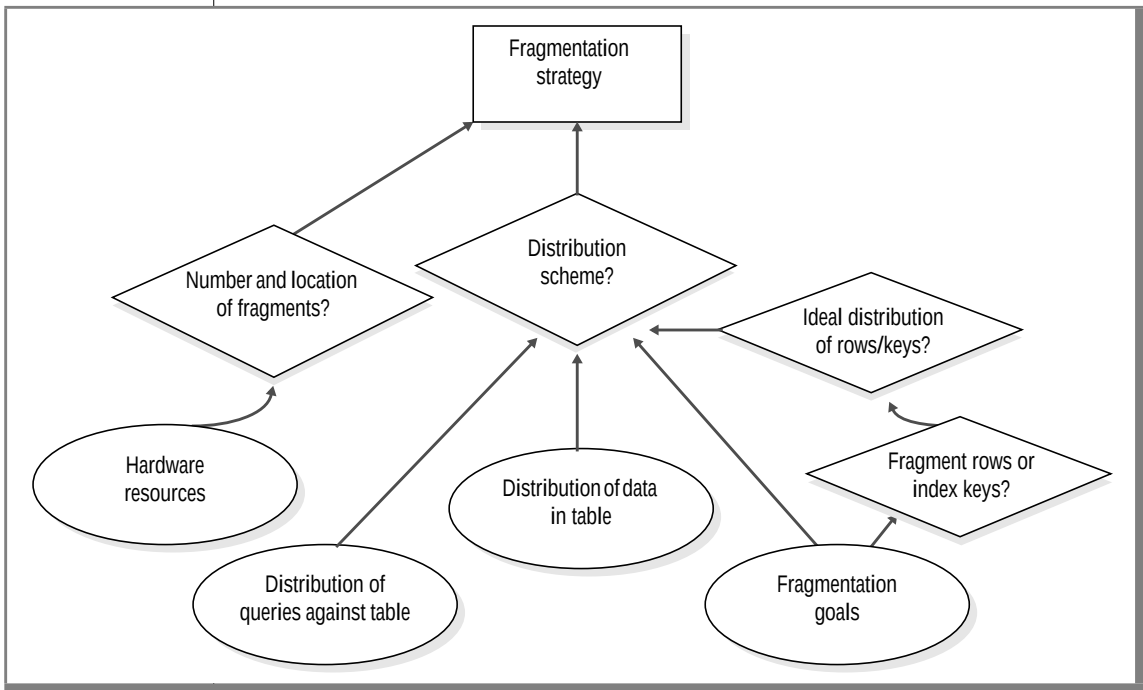
A fragmentation strategy consists of a distribution scheme and the set of dbspaces in which the fragments will reside. Formulating a fragmentation strategy requires you to make the following decisions:

1. Decide what your primary fragmentation goal is. Your fragmentation goals will 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 the ideal distribution of rows or index keys is for the table. This decision is also based on your primary fragmentation goal.

3. Decide on a distribution scheme. You must first choose between a round-robin and an expression-based distribution scheme. If you choose an expression-based distribution scheme, you must then design suitable fragment expressions.
4. To complete the fragmentation strategy, you must decide on the number and location of the fragments.

Figure 16-9 illustrates these decisions schematically. This section discusses the factors that influence these decisions.

Figure 16-9
Decisions and Information Necessary to Formulate a Fragmentation Strategy



Deciding How To Fragment a Table

Before you can come up with a fragmentation strategy, you must make certain fragmentation decisions. Specifically you must decide whether you need to fragment the table rows, the table index, or both, and how the table rows or index keys should be distributed. After you make these decisions, you can determine which distribution scheme best meets your needs.

Your fragmentation decisions usually depend on your fragmentation goal. The possible fragmentation goals are as follows:

- Improved single-user response time
- Improved concurrency
- Improved availability
- Improved backup-and-restore characteristics

Fragmenting for Single-User Response Time

The performance of an individual query can be improved if Universal Server processes the query in parallel using fragmentation with PDQ. For more information on PDQ, see [Chapter 18, “What Is PDQ?”](#) Improved single-user response time is usually the primary goal of fragmentation when queries against a table have long execution times. Typically, such queries are executed by decision-support applications that perform sequential reads of a large fraction of the rows in a table. For more information, see [“Decision-Support Applications” on page 18-7](#).

Fragment Table Rows or Index Keys?

Because improving single-user response time is usually the primary goal of fragmentation when queries against a table perform sequential scans of the data, you should fragment the table rows.

Do not fragment the table index if your applications are decision-support applications. If an index is fragmented, and a scan has to cross a fragment boundary for that index, the response time of the query is worse than if no fragmentation is used.



Tip: The previous argument assumes that the index scan is a serial scan. An application doing a long index scan can set the environment variables so that Universal Server performs a parallel index scan. In this case, a fragmented index does not cause response time to degrade. In general, however, index scans do not merit the cost in Universal Server resources required to perform parallel scans.

Ideal Distribution of Rows/Index Keys

When the primary goal of fragmentation is improved single-user response time, you should attempt to distribute all the rows of the table evenly over the different fragments. Overall query-completion time is reduced when Universal Server does not have to wait for the retrieval of data from a fragment that has more rows than other fragments.

Fragmenting for Improved Concurrency

Concurrency, in this context, refers to the ability of Universal Server to process multiple queries against a table simultaneously. Concurrency is important for tables that are heavily used because it provides a means of reducing contention for the data in that table. This scenario is typical when the applications that access the table are on-line transaction-processing (OLTP) applications. (For more information, see [“OLTP Applications” on page 18-6.](#))

Fragment Rows or Index Keys?

Concurrency is usually the primary goal of fragmentation when many simultaneous queries against a table perform index scans and return a relatively small number of rows. In this scenario, fragment both the index keys and rows using an expression-based distribution scheme that allows the queries to eliminate fragments from the scan. (See [“When Can the Database Server Eliminate Fragments from a Search?” on page 16-12.](#))

Ideal Distribution of Rows/Index Keys

The overall approach to reducing contention is as follows:

1. Investigate which queries access which parts of the table to be fragmented.
2. Fragment your data in such a way that some of the queries are routed to one fragment while others access a different fragment. Universal Server performs this routing when it evaluates the fragmentation rule for the table.
3. Store the fragments on separate devices.

The degree of success that you have in optimizing fragmentation for improved concurrency depends on how much you know about the distribution of data in the table, and the distribution of queries against the table. For example, if the distribution of queries against the table is such that all rows are accessed at roughly the same rate, you should attempt to distribute rows evenly across the fragments. On the other hand, if certain values are accessed at a higher rate than others, you can compensate for this situation by distributing the rows over the fragments unevenly.

Fragmenting for Improved Availability of Data

When you distribute table and index fragments across different disks or devices, you improve the availability of data during disk or device failures because Universal Server continues to allow access to fragments stored on disks or devices that are operational. This strategy has important implications for two types of applications:

- Applications that do not require access to unavailable fragments
A query that does not require Universal Server to access data in a fragment that is unavailable can still successfully retrieve data from fragments that *are* available. For example, if a table is fragmented by expression, and the expression is not complex, Universal Server can determine if a row is contained in a fragment without accessing the fragment. If the query accesses only rows that are contained in available fragments, the query succeeds even though some of the data in the table is unavailable.

- Applications that accept the unavailability of data

Some applications might be designed in such a way that they are able to accept the unavailability of data in a fragment and want the ability to retrieve the data that is available. These applications can specify which fragments can be skipped over by executing the SET DATASKIP statement before they execute a query. Alternatively, the Universal Server administrator can specify which fragments are unavailable with the **onspaces -f** option. For more information, refer to [“Skipping Inaccessible Fragments” on page 16-21](#).

Fragment Rows or Index Keys

If your fragmentation goal is increased availability of data, fragment both table rows and index keys. Fragment your rows and index keys using the same distribution scheme to ensure that the index keys reside on the same device as the rows that they index. If a disk drive fails, you can still access the data that resides on other devices.

Ideal Distribution of Rows/Index Keys

You cannot predict disk failures; therefore your data distribution should be motivated by other considerations than availability (for example, improved concurrency).

Fragmenting for Improved Backup-and-Restore Characteristics

If you use ON-Bar or ON-Archive for backup and recovery, both the unit of recovery and the Universal Server unit of storage for fragments is the dbspace. Consequently, you can define a unit of backup and recovery that is as small as several rows from a table or several key values from an index.

Fragment Rows or Index Keys

If your goal is finer granularity of backup and restore, you can fragment table data, table index, or both, depending upon what needs to be recovered rapidly in the event of a disk failure.

Ideal Distribution of Rows/Index Keys

When you develop a fragmentation strategy for finer granularity of backup and restore, consider the following suggestions:

- Keep the size of your fragments small.
Small fragments reduce recovery time if one of the fragments becomes corrupted.
- Group your rows carefully.
If a device that contains a dbspace fails, all the rows contained in the fragment of that dbspace are inaccessible. This consideration might influence which rows you group together in a particular fragment.

If you use ON-Bar for your backup and restore tool, refer to the [INFORMIX-Universal Server Backup and Restore Guide](#) for more information concerning backup and restore. If you use ON-Archive or **ontape** for your backup and restore tool, refer to the [INFORMIX-Universal Server Archive and Backup Guide](#).

Which Distribution Scheme Should You Use?

After you decide whether to fragment table rows or index keys, and how the rows or index keys should be distributed over fragments, you must decide on a distribution scheme to achieve this distribution of rows or index keys.

Choosing Round-Robin or Expression-Based Distribution Schemes

Figure 16-10 compares round-robin and expression-based distribution schemes in three important areas.

Figure 16-10
Key Considerations in Choosing a Distribution Scheme

Dist. Scheme	Ease of Data Balancing	Fragment Elimination	Data Skip
Round-robin	Automatic. In addition, Universal Server maintains data balancing over time.	Universal Server cannot eliminate fragments.	If data-skip feature is used, you cannot determine if the integrity of the transaction is compromised. However, you can always insert into a table fragmented by round-robin.
Expression based	Requires knowledge of the data distribution.	Depends on the fragment expressions. If nonoverlapping expressions on a single column are used, Universal Server can eliminate fragments for queries that have either range or equality expressions. See “When Can the Database Server Eliminate Fragments from a Search?” on page 16-12.	When data skip is used, you can determine whether the integrity of a transaction has been compromised. You cannot insert rows if the appropriate fragment for those rows is down.

The decision whether to use the round-robin distribution scheme or an expression-based distribution scheme is usually straightforward. Basically, round-robin provides the easiest and surest way of balancing data. However, with this distribution scheme, you do not have information on which fragment rows are located. This lack of information has negative consequences for the fragment-elimination characteristics. In general, therefore, round-robin is the correct choice only when *all* the following conditions apply:

- Your applications are decision-support applications.
- You *do not* know the distribution of data that will be added to the table.
- Your applications will not delete many rows (if they do, the load balancing could be degraded).

If the preceding conditions do not all apply, the appropriate distribution scheme is one of the expression-based distribution schemes. For example, if your applications are decision-support applications, but you *do* know what the data distribution is, fragment the data with an expression-based distribution scheme.

Designing an Expression-Based Distribution Scheme

The first step in designing a distribution scheme is to determine the distribution of data in the table, in particular the distribution of values for the column on which you are going to base the fragmentation expressions. To obtain this information, run `UPDATE STATISTICS` for the table and then examine the distribution with the **dbschema** utility. This distribution does not take into account data that will be *added* to the table. If you expect the table to be updated significantly, this distribution might be of limited use.

Once you know the data distribution, you can design a fragmentation rule that distributes data over the fragments as required to meet your fragmentation goal. For example, if your primary goal is to improve single-user response time, you can design fragment expressions that result in an even distribution of rows over the fragments.

If your primary fragmentation goal is improved concurrency, also analyze the queries that access the table. If certain rows are accessed at a higher rate than others, you can compensate by distributing the data over the fragments unevenly. See [“Fragmenting for Improved Concurrency” on page 16-26](#).

When you formulate the fragment expressions, try to create nonoverlapping expressions based on a single column with no `REMAINDER` fragment. This method will give you the best fragment-elimination behavior. (For a discussion of these guidelines, see [“When Can the Database Server Eliminate Fragments from a Search?” on page 16-12](#).) Fragment elimination is particularly important if your primary fragmentation goal is improved concurrency.

If you expect the size of the table to grow, but you cannot predict the data distribution for the table, consider using a hash rule to maintain an even distribution of rows or index keys. A trade-off is that if you use a hash rule, Universal Server cannot eliminate fragments for queries that have range expressions. For more information, see [“When Can the Database Server Eliminate Fragments from a Search?” on page 16-12](#).

How Many Fragments Should You Use?

After you choose a distribution scheme, the only task that remains in formulating a fragmentation strategy is to decide how many fragments to use and in which dbspaces to locate them.

Some upper limits exist on the number of fragments that you should use for your table, apart from the disk-storage resources you might have. These limits are related to your fragmentation goal.

The performance of decision-support queries increases linearly with the number of fragments added. However, the limit to this increase is determined by the number of CPUs and the bus bandwidth.

To obtain improved concurrency of OLTP queries, increase the number of fragments. However, as the fragments get smaller, the likelihood increases that range searches must scan multiple fragments and merge the results. This situation degrades the performance of these queries.

If your fragmentation goals are improved availability or finer granularity of backup and restore, consider the side effects that adding more fragments might have on performance and concurrency.

Managing Fragmentation

Fragmentation Tasks with SQL Statements	17-3
Skipping Fragments	17-4
Using the DATASKIP Configuration Parameter.	17-4
Using the SET DATASKIP Statement	17-5
Monitoring Fragmentation Use	17-6

Most tasks that you might want to perform require you to execute SQL statements. You should be perform these tasks in cooperation with the DBA. Figure 17-1 lists fragmentation tasks and the corresponding SQL statements required to perform the tasks. For more information about these statements, refer to the [Informix Guide to SQL: Tutorial](#) and the [Informix Guide to SQL: Syntax](#).

The following tasks are also discussed in this chapter:

- Skipping unavailable fragments
- Monitoring fragmentation use

Fragmentation Tasks with SQL Statements

You use appropriate SQL statements to perform most fragmentation tasks. Figure 17-1 lists common fragmentation tasks and the relevant SQL statements for these tasks. To obtain the syntax of these SQL statements, refer to the [Informix Guide to SQL: Syntax](#).

Figure 17-1
Fragmentation Tasks and Corresponding SQL Statement

Fragmentation Task	SQL Statement
Creating a new fragmented table	CREATE TABLE statement, FRAGMENT BY clause
Creating round-robin fragmentation for a smart large object column	CREATE TABLE statement, PUT clause
Creating a fragmented table from a single nonfragmented table	ALTER FRAGMENT statement, INIT FRAGMENT clause

(1 of 2)

Fragmentation Task	SQL Statement
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
Adding a fragment to a table	ALTER FRAGMENT statement, ADD clause
Dropping 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
Adding an explicit rowid column to a fragmented table	ALTER TABLE statement, ADD ROWID clause

(2 of 2)

Skipping Fragments

The Universal Server administrator can control whether unavailable fragments are skipped or not by setting the DATASKIP configuration parameter. Applications can override the DATASKIP parameter with the SET DATASKIP statement.

Using the DATASKIP Configuration Parameter

You can set the DATASKIP parameter to OFF, ALL, or ON *dbspace_list*. OFF means that Universal Server does not skip any fragments. If a fragment is unavailable, the query returns an error. ALL indicates that any unavailable fragment is skipped. ON *dbspace_list* instructs Universal Server to skip any fragments that are located in the specified dbspaces.

Use the **onspaces** utility to specify the dbspaces that are to be skipped when they are unavailable. For example, the following command sets the DATASKIP parameter so that Universal Server skips over the fragments in **dbspace1** and **dbspace3**, but not in **dbspace2**:

```
onspaces -f ON dbspace1 dbspace3
```

For the complete syntax for this **onspaces** option, see [“Specify DATASKIP Parameter” on page 39-74](#).

You can also use ON-Monitor to set the DATASKIP parameter. Select the **datasKip** option from the Dbspaces menu. You will see choices for setting DATASKIP to **Off** or to **On**, or to choose from a list of dbspaces. If you choose the list of dbspaces, a second screen appears that allows you to set DATASKIP for individual dbspaces.

Use the **onstat** utility to list the dbspaces currently affected by the **datasKip** feature. The **-f** option lists both the dbspaces that were set with the DATASKIP configuration parameter and the **-f** option of the **onspaces** utility. When you execute **onstat -f**, you see one of the following displays:

```
dataskip is OFF for all dbspaces
```

```
dataskip is ON for all dbspaces
```

```
dataskip is ON for dbspaces:  
dbspace1 dbspace2 ...
```

Using the SET DATASKIP Statement

An application can control whether a fragment should be skipped if it is unavailable by using the SQL statement SET DATASKIP. Applications should use this statement only in limited circumstances because it causes queries to return different results depending on the availability of the underlying fragments. Like the configuration parameter DATASKIP, the SET DATASKIP statement accepts a list of dbspaces that indicate to Universal Server which fragments to skip over. For example, suppose an application programmer included the following statement at the beginning of an application:

```
SET DATASKIP ON dbspace1, dbspace5
```

This statement causes Universal Server to skip over **dbspace1** or **dbspace5** whenever both of these conditions are met:

- The application attempts to access one of the dbspaces.
- Universal Server finds one of the dbspaces is unavailable.

If Universal Server finds that both **dbspace1** and **dbspace5** are unavailable, it skips both dbspaces.

The DEFAULT setting for the SET DATASKIP statement allows a Universal Server administrator to control the data-skip feature. Suppose an application developer includes the following statement in an application:

```
SET DATASKIP DEFAULT
```

When a query is executed subsequent to the preceding SQL statement, Universal Server checks the value of the configuration parameter DATASKIP. Encouraging end users to use this setting allows the Universal Server administrator to specify which dbspaces are to be skipped as soon as the Universal Server administrator becomes aware that one or more dbspaces are unavailable.

Monitoring Fragmentation Use

The Universal Server administrator might find the following aspects of fragmentation useful to monitor:

- Data distribution over fragments
- I/O request balancing over fragments
- The status of chunks that contain fragments

The Universal Server administrator can monitor the distribution of data over table fragments. If the goal of fragmentation is improved single-user response time, it is important for data to be distributed evenly over the fragments. To monitor fragmentation disk use, you must monitor Universal Server tblspaces because the unit of disk storage for a fragment is a tblspace. (For information on how to monitor the data distribution for a fragmented table, see [“Monitoring Tblspaces and Extents” on page 33-64.](#))

The Universal Server administrator must monitor I/O request queues for data contained in fragments. When I/O queues become unbalanced, the Universal Server administrator should work with the DBA to tune the fragmentation strategy. (For a discussion of how to monitor chunk use, including the I/O queues for each chunk, see [“Monitoring Chunks” on page 33-60.](#))

The Universal Server administrator must monitor fragments for availability and take appropriate steps when a dbspace that contains one or more fragments fails. For how to determine if a chunk is down, see [“Monitoring Chunk Status” on page 33-55.](#)

What Is PDQ?

What Is PDQ?	18-4
What Types of Applications Use Universal Server?	18-6
OLTP Applications	18-6
Decision-Support Applications	18-7
When Should You Use PDQ?	18-8
Processing OLTP Queries	18-9
Processing Decision-Support Queries	18-10
How Does the Database Server Allocate Resources with PDQ?	18-11
Parameters Used for Controlling PDQ	18-11
PDQ Priority	18-12
MAX_PDQPRIORITY	18-13
DS_MAX_QUERIES	18-13
DS_MAX_SCANS	18-13
DS_TOTAL_MEMORY	18-14
OPTCOMPIND	18-14
How Does the Database Server Use PDQ?	18-14
SQL Operations That Take Advantage of PDQ	18-14
Parallel Inserts.	18-15
Parallel Index Builds	18-17
SQL Operations That Do Not Use PDQ	18-17
Update Statistics and PDQ	18-18
Stored Procedures, Triggers, and PDQ	18-18
Correlated and Uncorrelated Subqueries and PDQ	18-19
Outer Index Joins and PDQ	18-19
Remote Tables and PDQ.	18-19

How Does the Database Server Manage PDQ Queries?	18-20
How Does PDQ Interact with Parallel Sorts?	18-21
Sorting When PDQ Priority Is Zero	18-21
Sorting When PDQ Priority Is Greater Than Zero	18-21

Parallel database query (PDQ) is an INFORMIX-Universal Server feature that can improve performance dramatically when Universal Server processes queries initiated by decision-support applications. The features of PDQ allow Universal Server to distribute the work for one aspect of a query among several processors. For example, if a query requires an aggregation, Universal Server can distribute the work for the aggregation among several processors. PDQ also includes tools for resource management. This chapter and [Chapter 19, “Managing PDQ and Decision Support,”](#) describe how PDQ works and how to use the parameters that control PDQ.

Another Universal Server feature, *table fragmentation*, allows you to store the parts of a table on different disks. PDQ delivers maximum performance benefits when the data that is being queried is contained in fragmented tables. [Chapter 16, “What Is Fragmentation?”](#) describes fragmented tables and how to use PDQ and fragmentation together for maximum performance.

PDQ is not appropriate for all queries. In fact, you will see serious performance degradation if you use PDQ inappropriately. This chapter describes the situations in which PDQ is useful and tells how to activate PDQ. The chapter also describes the tools that you use to allocate resources among PDQ and non-PDQ queries.

What Is PDQ?

PDQ refers to the techniques that Universal Server can use to distribute the execution of a single query over several processors. Universal Server can also use PDQ for queries that consume large quantities of non-CPU resources, in particular large quantities of memory and disk scans.

A query processed with PDQ techniques is called a *PDQ query*. When Universal Server processes a PDQ query, it first subdivides the query into subqueries. Universal Server then allocates the subqueries to a number of threads that process the subqueries in parallel. Because each subquery represents a smaller amount of processing time when compared to the original query, and because each subquery is processed simultaneously with all other subqueries, Universal Server is able to drastically reduce the time required to process the query. Figure 18-1 illustrates this concept.

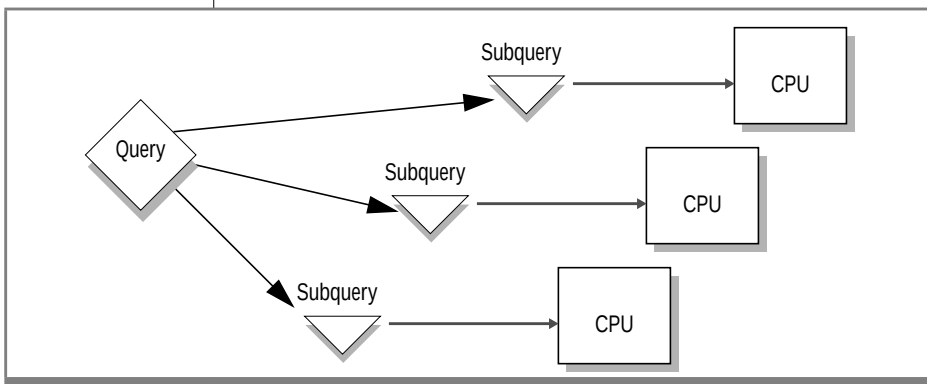


Figure 18-1
Parallel Database Query

PDQ has five principal components, as follows:

- Parallel scan
- Parallel join
- Parallel sort
- Parallel aggregate
- Parallel group

The degree of parallelism for a query refers to the number of subqueries that the database server will execute in parallel to run the query. For example, a two-table join that is executed by six threads (with each thread executing one-sixth of the required processing) has a higher degree of parallelism than one executed by two threads.

Universal Server determines the best degree of parallelism for each component of a PDQ query, based on values set by the Universal Server 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, Universal Server uses the components themselves in parallel. For example, consider what occurs when Universal Server must process a complex join. First, Universal 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, Universal Server begins the sort, and so on, until the full join is completed.

When PDQ is used appropriately, it provides performance advantages on both multiprocessor and uniprocessor computers. On a multiprocessor computer, PDQ distributes the execution of a query across available processors. On a uniprocessor computer, PDQ allows Universal Server to submit I/O requests to multiple disks in parallel and to take full advantage of the memory on the computer. If you wish to derive the full performance benefits that PDQ can offer, Informix recommends that you run PDQ on a multiprocessor computer.

You gain the most benefit from PDQ when you use fragmented tables on a multiprocessor computer. However, PDQ can give you performance gains even with nonfragmented tables on a uniprocessor computer.

What Types of Applications Use Universal Server?

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. Do not use PDQ for OLTP queries. 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 are typically multiuser and response times are measured in fractions of seconds.

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

Decision-Support Applications

Decision-support applications provide information used for strategic planning, decision making, and report preparation. These applications are frequently executed in a batch environment. Typical applications include payroll, inventory reporting, and end-of-period accounting reports. Figure 18-2 illustrates the difference between OLTP and decision-support applications.

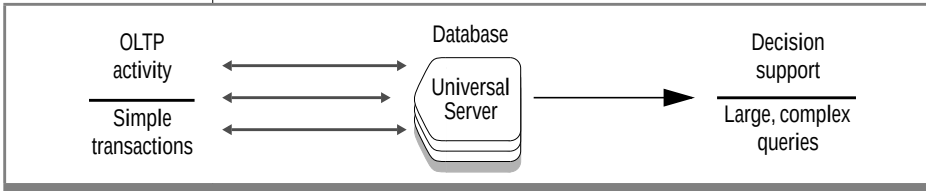


Figure 18-2
*OLTP Operations
Contrasted with
Decision-Support
Operations*

Decision-support applications perform tasks that are more complex than the simple reads and writes that characterize OLTP. Decision-support applications frequently generate queries that require Universal Server to scan entire tables and manipulate large amounts of data. These queries require operations such as multiple joins, temporary tables, and hundreds, if not thousands, of calculations. Such operations involve 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.

Decision-support queries consume large quantities of non-CPU resources and, in particular, large quantities of memory. Universal Server typically allocates large quantities of memory to queries that contain one or more of the following operations:

- Merge joins
- Hash joins
- Sorting
- Groups

In addition, other factors influence how Universal 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 Universal Server must allocate memory and temporary disk space to sort the query. However, if column **col1** is indexed, Universal Server can sometimes avoid performing the sort by scanning the table using the index and can automatically provide the ordering requested by the user. If no sort is required, the query does not consume non-CPU resources.

In summary, 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

Decision-support applications are large consumers of computing resources. If the results of decision-support applications are required during working hours, decision-support and OLTP applications contend for computing resources, resulting in unacceptable performance degradation for both. The features of PDQ provide a solution to many of the problems posed when you are running decision-support applications and OLTP applications on the same computer.

When Should You Use PDQ?

Universal Server uses as many computer resources as possible to complete a query. These resources include memory, CPU, and disk I/O. When both OLTP and decision-support queries are running on the same computer, Universal Server must balance its resources so that all users receive the best possible performance. Universal Server can process decision-support queries more efficiently using PDQ, but, if unchecked, these queries could absorb the entire resources of a computer for hours. OLTP queries cannot perform satisfactorily in such an environment. Therefore, Universal Server in effect maintains two sets of priorities, one for OLTP and one for decision support.

Universal Server uses the *PDQ priority* value of a query to determine when to use PDQ to process a query in parallel. The default value of PDQ priority for all applications is 0, which means that PDQ processing is not used. Users request PDQ processing and a percentage of PDQ resources when they set the PDQ priority value greater than 0. For more information on PDQ priority, refer to “[PDQ Priority](#)” on page 18-12.

Processing OLTP Queries

By default, Universal Server optimizes performance for short transactions that require rapid response times. In other words, the default behavior of Universal Server is appropriate for OLTP transactions. All queries have the same priority for CPU, memory, and disk I/O. To obtain this default behavior, users can set the **PDQPRIORITY** environment variable to 0 before they run their applications.



Warning: Users must not set the **PDQPRIORITY** environment variable to a non-zero value for OLTP queries. That is, do not allow users to process OLTP queries as PDQ queries.

Consider the following situation. One of the parameters that the Universal Server administrator can set to control PDQ queries limits the number of simultaneous queries. Suppose the number of simultaneous queries is set to 4. Four PDQ queries might start, each requiring about 25 minutes to complete. If another PDQ query requests service, it cannot start. The fifth query waits until one of the previous four queries finishes. OLTP queries cannot wait 25 minutes for service; they must be processed immediately.

Queries that should *not* use PDQ require quick response and generate only a small amount of information. For example, the following queries should not use the PDQ features of Universal Server:

- 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?

Processing Decision-Support Queries

If a user sets the PDQ priority greater than 0 for a given query, Universal 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. Queries that should use PDQ are decision-support type queries. For example, the following questions should use the PDQ features of Universal 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?

Universal Server determines the degree to 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 values of Universal Server system-wide parameters (NUMCPUVPS, DS_TOTAL_MEMORY, and so forth) set by the Universal Server administrator

The parameters that control PDQ resources are discussed in the following section.

How Does the Database Server Allocate Resources with PDQ?

Without resource management, the performance of decision-support applications could be very uneven, and OLTP applications and other applications that share the resources of the computer could suffer degradation. This section discusses the tools that you can use to balance resource use.

Parameters Used for Controlling PDQ

Figure 18-3 summarizes the configuration parameters, environment variables, and the SQL statement used by Universal Server to control how resources are allocated 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 18-3
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 Universal 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

PDQ Priority

Users request what part of Universal 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.

A small value for PDQ priority gives PDQ processing a low priority (few resources), and a high value gives a high priority (many resources).

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 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. In either case, the Universal Server administrator can limit the percentage of PDQ resources that Universal Server actually allocates to users by setting the MAX_PDQPRIORITY configuration parameter.

The default value of PDQ priority for all applications is 0. Therefore, users must explicitly set PDQ priority to execute a query in parallel.

The environment variable and the SQL statement use the same range of values for setting PDQ priority. In addition, the DEFAULT tag 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: Universal 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.

MAX_PDQPRIORITY

The MAX_PDQPRIORITY configuration parameter limits the amount of resources that PDQ query can request. Thus, the Universal Server administrator can limit the resources used by PDQ, even though an application or a user might request a higher degree of PDQ use.

For more information on the configuration parameter MAX_PDQPRIORITY, refer to [“MAX_PDQPRIORITY” on page 37-43](#). For more information on the performance implications of MAX_PDQPRIORITY, refer to the [INFORMIX-Universal Server Performance Guide](#).

DS_MAX_QUERIES

The DS_MAX_QUERIES configuration parameter specifies the maximum number of queries that can run concurrently. If, as specified in DS_MAX_QUERIES, the maximum number of queries is already running, an application that tries to start a PDQ query is blocked until one of the earlier queries finishes. For information about setting DS_MAX_QUERIES, refer to [“DS_MAX_QUERIES” on page 37-21](#).

DS_MAX_SCANS

The DS_MAX_SCANS configuration parameter limits the number of concurrent decision-support scans that Universal Server allows. Setting DS_MAX_SCANS to an appropriate value prevents flooding Universal Server with an excess of scan requests. For information about setting DS_MAX_SCANS, refer to [“DS_MAX_SCANS” on page 37-22](#).

DS_TOTAL_MEMORY

The DS_TOTAL_MEMORY configuration parameter specifies the amount of memory available for decision-support queries. By limiting the amount of memory available for PDQ, you can control how much of the system resources the decision-support queries can use. For information about choosing a value for DS_TOTAL_MEMORY, refer to [“Adjusting the Amount of Memory” on page 19-7](#).

OPTCOMPIND

The OPTCOMPIND configuration parameter and the **OPTCOMPIND** environment variable affect the behavior of queries, including PDQ queries, but do not control PDQ resources. OPTCOMPIND suggests an appropriate join method to the optimizer. Different join methods have advantages and disadvantages that depend on the type of the query, the indexes available, and the isolation level of the query. If you know in advance what type of joins would be advantageous, you can help the optimizer decide how to execute the queries. For more information about choosing a value for OPTCOMPIND, refer to [“Using OPTCOMPIND” on page 19-10](#).

How Does the Database Server Use PDQ?

This section describes the types of SQL operations that Universal Server processes in parallel and the situations that limit the degree of parallelism that Universal Server can use. In the following discussions, a *query* is any SELECT statement.

SQL Operations That Take Advantage of PDQ

Universal Server can potentially use PDQ to process any query in parallel. How Universal Server processes queries in parallel is described in [“What Is PDQ?” on page 18-4](#).

Universal Server treats the DELETE, INSERT, and UPDATE statements as two-step statements, as follows:

1. Fetch the qualifying rows.
2. Apply the action of deleting, inserting, or updating.

Universal Server processes the first step of the statement in parallel, with one exception; Universal 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.

The following sections describe other SQL operations that take advantage of PDQ processing:

- Parallel inserts
- Parallel index builds

Parallel Inserts

Universal 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

Chapter 14, “Where Is Data Stored?” and the *Informix Guide to SQL: Syntax* discuss implicit and explicit temporary tables.

The following sections explain the details and restrictions that apply to parallel inserts.

Explicit Inserts Using SELECT...INTO TEMP

Universal Server can insert rows in parallel into explicit temporary tables that you specify in SQL statements of the form SELECT...INTO TEMP. For example, Universal 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
```

Universal 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 wish Universal 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 Universal Server performs the insert. To perform the insert in parallel, Universal Server first creates a fragmented temporary table. So that Universal 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 **DBSPAC-ETEMP** environment variable. In addition, you must set DBSPACETEMP to indicate storage space for the fragments *before* you execute the SELECT...INTO statement.

Universal 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

Universal 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, Universal 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.

Universal 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.

Universal Server does not process parallel inserts that reference a stored procedure. For example, Universal 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. Universal 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 18-11](#).

If you have a computer with multiple CPUs, Universal Server uses two sort threads to sort the index keys. Universal Server uses two sort threads during index builds without the user setting the **PSORT_NPROCS** environment variable.

SQL Operations That Do Not Use PDQ

Universal 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 and PDQ

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, Universal Server must allocate the memory that this statement uses for sorting.

Stored Procedures, Triggers, and PDQ

Statements that involve stored procedures are not executed in parallel. However, statements within procedures are executed in parallel.

When Universal 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 Universal Server. For a complete list of DML statement, see the [Informix Guide to SQL: Syntax](#).

Universal 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 and PDQ

Universal 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 and PDQ

Universal 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, Universal Server lowers the PDQ priority of only that subquery or view, not of the parent query or any other subquery.

Remote Tables and PDQ

A remote table has all its fragments in the same nonlocal database. Although Universal Server can process the data stored in a remote table in parallel, the data is communicated serially because Universal Server allocates a single thread to submit and receive the data from the remote table.

Universal 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.

How Does the Database Server Manage PDQ Queries?

The *memory grant manager* (MGM) manages the allocation of resources used by PDQ. MGM controls the amount of memory and the number of I/O scan threads allocated to each query. MGM uses configuration parameters and one environment variable to manage PDQ. The PDQ configuration parameters are DS_MAX_QUERIES, DS_TOTAL_MEMORY, DS_MAX_SCANS, and MAX_PDQPRIORITY. The environment variable is **PDQPRIORITY**.

The Universal Server administrator can influence the decisions of MGM by adjusting these PDQ parameters in the ONCONFIG file or by using the **onmode** utility to change their values while Universal Server is in on-line mode. The settings for these configuration parameters appear in [Chapter 37, “Configuration Parameters.”](#) For instructions on using **onmode**, see [“Change Decision-Support Parameters” on page 39-46.](#)

When Universal Server receives a PDQ query, it places that query in a *priority queue*. All PDQ queries wait in this queue until Universal Server has resources available to service the request. The queue is a *priority queue* because the query with the highest PDQ priority receives service first. MGM performs the following tests to see if it has enough resources to allow the highest priority request to run:

- Is there any memory available? If not, wait for memory.
- Are DS_MAX_SCANS scan threads already running? If so, wait until a scan thread becomes available.
- Has a higher priority query entered the queue? If so, go back into the queue.
- Are DS_MAX_QUERIES queries already running? If so, wait until a query finishes.
- Is MGM in the process of initializing because you used the **onmode** utility to change the value of a PDQ configuration parameter? If so, wait until initialization is finished.

The tests that a query must pass before it receives PDQ resources are sometimes referred to as *gates*. The query must pass through each gate before it receives any resources.

Application writers can use the SQL statement `SET PDQPRIORITY` to modify the priorities of an application. Application users can modify the resources they receive by setting their **PDQPRIORITY** environment variable. The `SET PDQPRIORITY` statement is documented in the [Informix Guide to SQL: Syntax](#). The **PDQPRIORITY** environment variable is documented in the [Informix Guide to SQL: Reference](#).

How Does PDQ Interact with Parallel Sorts?

Universal Server can use parallel sorts for any query; parallel sorts are not limited to PDQ queries. The **PSORT_NPROCS** environment variable specifies the maximum number of threads that can be used to sort a query.

Sorting When PDQ Priority Is Zero

Universal Server *does not* control sorting by any of the PDQ configuration parameters when PDQ priority is 0. When the value of PDQ priority is 0, and **PSORT_NPROCS** is greater than 1, Universal Server usually does not use parallel sorts. When PDQ priority is 0, the maximum amount of shared memory that Universal Server allocates for a sort is about 128 kilobytes. This amount of sort memory is usually not enough for parallel sorting.

Sorting When PDQ Priority Is Greater Than Zero

When PDQ priority is greater than 0, and **PSORT_NPROCS** is greater than 1, SQL statements with sort operations, such as queries and index builds, benefit both from parallel sorts and from PDQ features such as parallel scans and additional memory. For more information on how memory is allocated to PDQ queries for sorts, refer to [“Sorting Memory” on page 12-32](#).

When you are running PDQ queries, Informix recommends that you initially set **PSORT_NPROCS** to 2. Then monitor the I/O and CPU activity. If the CPU activity is slower than the I/O activity, you can increase **PSORT_NPROCS**.

For more information on parallel sorts and the **PSORT_NPROCS** environment variable, refer to [Chapter 12, “Shared Memory,”](#) in the [INFORMIX-Universal Server Performance Guide](#).

Managing PDQ and Decision Support

Controlling the Use of Resources	19-3
Limiting the Priority of PDQ Queries	19-4
Setting MAX_PDQPRIORITY	19-5
Maximizing OLTP Throughput	19-6
Conserving Resources	19-6
Allowing Maximum Use of Parallelism	19-6
Using Stored Procedures	19-6
Adjusting the Amount of Memory	19-7
Limiting the Number of Concurrent Scans	19-8
Limiting the Maximum Number of Queries	19-9
Managing Applications	19-9
Using SET EXPLAIN	19-9
Using OPTCOMPIND	19-10
Using SET PDQPRIORITY	19-10
End-User Control of Resources	19-11
Database Administrator Control of Resources	19-11
Using the Memory Grant Manager	19-12

T

his chapter discusses the parameters and strategies that you use to manage PDQ. The chapter addresses both database administrators and end users who want to take advantage of the performance benefits provided by PDQ. The chapter covers the following topics:

- Controlling the use of resources
- Balancing resource requirements for OLTP and decision-support applications
- Using the memory grant manager
- Monitoring PDQ

Controlling the Use of Resources

When the Universal Server SQL query executor uses PDQ to execute a query in parallel, it puts a heavy load on the operating system. In particular, PDQ controls the use of the following resources:

- Memory
- CPU VPs
- Disk I/O (temporary table space, fragmented tables)
- Scan threads

When you configure Universal Server, consider how the use of PDQ affects users of OLTP, users of decision-support applications, and users of the computer who are not Universal Server users.

You can control how Universal Server uses resources in the following ways:

- Limit the priority of PDQ queries.
- Adjust the amount of memory.

- Limit the number of scans threads.
- Limit the number of concurrent queries.

If your applications use temporary tables or large sort operations, you can improve performance by designating one or more dbspaces for temporary tables and sort files using the `DBSPACETEMP` configuration parameter or the **DBSPACETEMP** environment variable. For more information on configuring dbspaces for temporary tables and sorting, refer to Chapter 2 “Configuration Impacts on Performance” in the [INFORMIX-Universal Server Performance Guide](#).

Limiting the Priority of PDQ Queries

The default value for the priority of individual applications is 0, which means that PDQ processing is not used. Universal Server does not use PDQ unless it is turned on by one of the following events:

- A user sets the **PDQPRIORITY** environment variable.
- A `SET PDQPRIORITY` statement appears in an application.

The **PDQPRIORITY** environment variable and the `MAX_PDQPRIORITY` configuration parameter work together to control the amount of resources to allocate for parallel processing. Setting this environment variable and configuration parameter correctly is critical for the effective operation of PDQ.

The `MAX_PDQPRIORITY` configuration parameter allows the Universal Server administrator to limit the parallel-processing resources consumed by PDQ queries. Thus, the **PDQPRIORITY** environment variable sets a *reasonable* or *recommended* priority value, while `MAX_PDQPRIORITY` limits the priority that an application can claim.

User **informix** or user **root** can use the **onmode** command-line utility to change the values of the configuration parameters that manage PDQ resources while Universal Server is in on-line mode, as follows:

- Use **onmode -M** to change the value of `DS_TOTAL_MEMORY`.
- Use **onmode -Q** to change the value of `DS_MAX_QUERIES`.
- Use **onmode -D** to change the value of `MAX_PDQPRIORITY`.
- Use **onmode -S** to change the value of `DS_MAX_SCANS`.

For more information about using **onmode**, refer to [“Change Decision-Support Parameters” on page 39-46](#).

Changes to the PDQ parameters made with **onmode** are not permanent. When Universal Server is reinitialized, the PDQ parameters in the ONCONFIG configuration file are used.

If you must change the values of the decision-support parameters on a regular basis (for example, to set MAX_PDQPRIORITY to 100 each night for processing reports), you can set the values using a scheduled operating-system job. For information about creating scheduled jobs, refer to your operating-system manuals.

To obtain the best performance from your Universal Server database server, you must choose values for your parameters and then observe the behavior of Universal Server and adjust these values. No well-defined rules exist for choosing parameter values.

Setting MAX_PDQPRIORITY

The MAX_PDQPRIORITY configuration parameter limits the value of PDQ priority that Universal Server grants any one decision-support query. When an application or an end user attempts to set a PDQ priority, the priority granted is multiplied by the percentage specified in MAX_PDQPRIORITY.

The possible range of values for MAX_PDQPRIORITY is 0 to 100. This range represents the percent of resources that you can allocate to processing a given decision-support query. Reduce the value of MAX_PDQPRIORITY to 0 or 1 to allocate more resources to OLTP queries or other decision-support queries. Increase the value of MAX_PDQPRIORITY when you want to allocate more resources to decision support.

Maximizing OLTP Throughput

At times, you might want to allocate resources to maximize the throughput of individual OLTP queries rather than allocating resources for decision support and intraquery parallelism. One way to maximize OLTP throughput is to reduce DS_TOTAL_MEMORY. Another way is to set MAX_PDQPRIORITY to 0, which effectively turns off PDQ priority. Turning off PDQ priority does not prevent decision-support queries from running; it just does not allow decision-support queries to use resources in parallel. With PDQ priority off or with less DS_TOTAL_MEMORY, response for decision-support queries can be very slow.

Conserving Resources

If end-user applications make little use of queries that would require parallel sorts, parallel group-by, or parallel joins, consider using the LOW setting.

If Universal Server is operating in a multiuser environment, you might set MAX_PDQPRIORITY to 1 to increase interquery performance at the cost of some intraquery parallelism. A trade-off exists between these two different types of parallelism because they are competing for the same resources. As a compromise, you might set MAX_PDQPRIORITY to some intermediate value (perhaps 20 or 30).

Allowing Maximum Use of Parallelism

Set MAX_PDQPRIORITY to 100 if you want Universal Server to assign as many resources as possible to parallel processing. This setting is appropriate for times when parallel processing will not interfere with OLTP processing or with other (non-Universal Server) users of your computer.

Using Stored Procedures

Universal Server freezes the PDQ priority used to optimize SQL statements within procedures at the time of procedure creation or the last manual recompilation with the UPDATE STATISTICS statement.

To change the client value of PDQ priority, embed the SET PDQPRIORITY statement within the body of your procedure. If no such statement has been executed, the value that was in effect when the procedure was last compiled or created is used. The PDQ priority currently in effect outside a procedure is ignored within a procedure.

Informix suggests that you turn PDQ priority off when you enter a procedure and then turn it on again for specific statements. You avoid tying up large amounts of memory for the procedure, and you can ensure that the crucial parts of the procedure use the appropriate PDQ priority, as the following example illustrates:

```
CREATE PROCEDURE my_proc (a INT, b INT, c INT)
    Returning INT, INT, INT;
SET PDQPRIORITY 0;
.
.
.
SET PDQPRIORITY 85;
SELECT ..... (big complicated SELECT statement)
SET PDQPRIORITY 0;
.
.
.
;
```

Adjusting the Amount of Memory

Use the following formula as a starting point for estimating the amount of shared memory to allocate to decision-support queries:

$$\text{DS\_TOTAL\_MEMORY} = p\_mem - os\_mem - rsdnt\_mem - (128K * users) - other\_mem$$

The variables in the formula are defined as follows:

<i>p_mem</i>	total physical memory available on host
<i>os_mem</i>	size of operating system including buffer cache
<i>resdnt_mem</i>	size of Informix resident shared memory
<i>users</i>	number of expected users (connections) specified in the NETTYPE configuration parameter
<i>other_mem</i>	size of memory used for other (non-Informix) applications

The value for DS_TOTAL_MEMORY derived from this formula should serve only as a starting point. To arrive at a value that makes sense for your configuration, you must monitor paging and swapping. (Use the tools provided with your operating system to monitor paging and swapping.) When paging increases, decrease the value of DS_TOTAL_MEMORY.

The amount of memory granted to a single PDQ query is influenced by many system factors, but, in general, the memory granted to a single PDQ query is proportional to the following formula:

$$\text{memory\_grant} = \text{DS\_TOTAL\_MEMORY} * (\text{PDQPRIORITY}/100) * (\text{MAX\_PDQPRIORITY} / 100)$$

The lower size limit of a memory grant is 16 kilobytes or 1 memory-grant increment, whichever is greater.

Limiting the Number of Concurrent Scans

Universal Server apportions some number of scans to a query according to its PDQ priority (among other factors). DS_MAX_SCANS and MAX_PDQPRIORITY allow you to limit the number of PDQ scan threads that can run concurrently.

For example, suppose a very large table contains 100 fragments. With no limit on the number of concurrent scans allowed, Universal Server would concurrently execute 100 scan threads to read this table. In addition, as many end users as wished to could initiate this query.

As the Universal Server administrator, you set DS_MAX_SCANS to a value lower than the number of fragments in this table to prevent Universal Server from being flooded with scan threads by multiple decision-support queries. You can set DS_MAX_SCANS to 20 to ensure that Universal Server concurrently executes a maximum of 20 scan threads for parallel scans. Furthermore, if multiple users initiate PDQ queries, each query receives only a percentage of the 20 scan threads, according to the PDQ priority assigned to the query and the MAX_PDQPRIORITY that the Universal Server administrator sets.

For more information on DS_MAX_SCANS, refer to [“DS_MAX_SCANS” on page 37-22](#).

Limiting the Maximum Number of Queries

The DS_MAX_QUERIES configuration parameter limits the number of concurrent decision-support queries that can run. Estimate the number of decision-support queries that Universal Server can run concurrently and set DS_MAX_QUERIES accordingly.

For more information on DS_MAX_QUERIES, refer to [“DS_MAX_QUERIES” on page 37-21](#).

Managing Applications

The Universal Server administrator, the writer of an application, and the end user all have a certain amount of control in determining the amount of resources that Universal Server allocates to processing a query. The Universal Server administrator exerts control through the use of configuration parameters. The application developer or the end user can exert control through the use of an environment variable or through an SQL statement.

Using SET EXPLAIN

The output of the SQL SET EXPLAIN statement shows decisions made by the query optimizer. It shows whether parallel scans are used, the maximum number of threads required by the query, and the type of join used for the query. You can use SET EXPLAIN to study the query plans of an application. You can restructure a query or use OPTCOMPIND to change how the optimizer treats the query.

Using OPTCOMPIND

The **OPTCOMPIND** environment variable and the OPTCOMPIND configuration parameter indicate the preferred join method, thus assisting the optimizer in selecting the appropriate join method for parallel database queries.

To influence the optimizer in its choice of a join method, you can set the OPTCOMPIND configuration parameter. Universal Server uses the value of this configuration parameter only if the user of the application has not set the **OPTCOMPIND** environment variable.

When you set OPTCOMPIND, keep in mind the nature of hash joins and sort-merge joins. An application with isolation mode set to Repeatable Read can lock all records in tables that are involved in the hash join or sort-merge join. For this reason, Informix recommends that you set OPTCOMPIND to 1. For more information on the different join methods, refer to Chapter 4 of the *INFORMIX-Universal Server Performance Guide*.

If you want the optimizer to make the determination for you based on costs, regardless of the isolation mode setting of end-user applications, set OPTCOMPIND to 2.

For more information on the OPTCOMPIND configuration parameter, refer to “[OPTCOMPIND](#)” on page 37-54. For more information about data distributions, see the UPDATE STATISTICS statement in the *Informix Guide to SQL: Syntax*.

Using SET PDQPRIORITY

The SET PDQPRIORITY statement allows an application to set PDQ priority dynamically within an application. The PDQ priority value can be any integer from -1 through 100.

The PDQ priority set with the SET PDQPRIORITY statement supersedes the environment variable **PDQPRIORITY**.

End-User Control of Resources

An end user indicates the PDQ priority of a query by setting the **PDQPRIORITY** environment variable or by executing the SQL statement **SET PDQPRIORITY** prior to issuing a query. In effect, end users can request a certain amount of parallel-processing resources for the query.

The amount of resources that the user requests and the amount that Universal Server actually allocates for the query can be different. This situation occurs when the Universal Server administrator uses the **MAX_PDQPRIORITY** configuration parameter to limit user-requested resources. For example, if a client uses the **SET PDQPRIORITY 80** statement to request 80 percent of PDQ resources, but **MAX_PDQPRIORITY** is set to 50, the client is allocated only 40 percent of the resources (50 percent of the request).

Database Administrator Control of Resources

As Universal Server administrator, you can influence the amount of overall resources that Universal Server allocates to processing queries in parallel.

You can manipulate **MAX_PDQPRIORITY** to control the amount of resources that Universal Server allocates to OLTP and the amount that it allocates to decision support. For example, if a period during the day has particularly heavy OLTP processing, you can set **MAX_PDQPRIORITY** to 0. This configuration parameter limits the PDQ priority assigned by end users or by the application. When **MAX_PDQPRIORITY** is 0, PDQ processing is turned off until you reset **MAX_PDQPRIORITY** to a nonzero value.

In addition to **MAX_PDQPRIORITY**, you can control the resources that Universal Server allocates to decision-support queries by setting the **DS_TOTAL_MEMORY**, **DS_MAX_SCANS**, and **DS_MAX_QUERIES** configuration parameters. In addition to setting limits for decision-support memory and the number of decision-support queries that can run concurrently, Universal Server uses these parameters to determine the amount of memory to allocate to individual decision-support queries as they are submitted by users.

You can also limit the number of concurrent decision-support scans that Universal Server allows by setting the **DS_MAX_SCANS** configuration parameter.

Previous versions of Universal Server allowed you to set a PDQ priority configuration parameter in the ONCONFIG file. If your applications depend on a global setting for PDQ priority, you can obtain a similar functionality by including **PDQPRIORITY** as a shared environment variable in the **informix.rc** file. For more information on the **informix.rc** file, see the [Informix Guide to SQL: Reference](#).

Using the Memory Grant Manager

The memory-grant manager (MGM) is a Universal Server component that coordinates the use of CPU, memory and scan threads for decision-support queries. MGM uses the DS_MAX_QUERIES, DS_TOTAL_MEMORY, DS_MAX_SCANS, and MAX_PDQPRIORITY configuration parameters, and the **PDQPRIORITY** environment variable to determine what resources can or cannot be granted to a decision-support query.

When your Universal Server instance has heavy OLTP use, and you find performance is degrading, you can use MGM monitoring tools to monitor the memory committed to PDQ queries. If your decision-support queries are consuming excessive quantities of memory, you can decrease the memory committed to PDQ queries or limit the number of scans. During off-peak hours, you can designate a larger proportion of the resources to parallel processing to achieve high throughput on decision-support queries.

To monitor how MGM coordinates memory usage and scan threads, execute the **onstat -g mgm** option. This option reads shared-memory structures and provides statistics that are accurate at the instant that the command executes. For an example of the **onstat -g mgm** output and other **onstat** options that you can use to obtain information on decision-support threads, see [“Monitoring Parallel Database Query” on page 33-40](#).

Logging and Log Administration

Section V



What Is Logging?

Which Database Server Processes Require Logging?	20-3
What Database Server Activity Is Logged?	20-5
Activity That Is Always Logged	20-6
Activity Logged for Databases with Transaction Logging	20-7
Are Simple Large Objects Logged?	20-7
Are Smart Large Objects Logged?	20-8
When Logging Begins	20-8
What Is Transaction Logging?	20-9
The Database-Logging Status	20-10
Unbuffered Transaction Logging	20-10
Buffered Transaction Logging	20-11
ANSI-Compliant Transaction Logging	20-11
Databases with Different Log-Buffering Status.	20-11
When to Use Transaction Logging	20-12
When to Buffer Transaction Logging	20-12
Who Can Set or Change Logging Status	20-13

T

his chapter describes the functionality of INFORMIX-Universal Server logging. First, the chapter describes logging with respect to Universal Server functionality. It addresses the following questions:

- Which Universal Server features require logging?
- What Universal 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 Universal 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, Universal Server makes no further use of the log records. However, when Universal 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 Universal Server data-recovery mechanisms.

Which Database Server Processes Require Logging?

Universal Server stores the log records in a *logical log*. The logical log is made up of *logical-log files* that Universal Server manages on disk until they have been safely transferred off-line (*backed up*). The Universal 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. [Chapter 22, “What Is the Logical Log?”](#), explains logical-log administration topics.

Universal Server uses logical-log records when it performs various functions that recover data and ensure data consistency, as follows:

- Fast recovery

If Universal Server shuts down in an uncontrolled manner, Universal 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. Universal 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 26-5](#).)

- Transactions roll back

If a database has transaction logging turned on (see [“What Is Transaction Logging?” on page 20-9](#)), and a transaction must be rolled back, Universal 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 Universal Server dbspace backup tapes to re-create the Universal Server system up to the point of the most-recently backed-up logical-log record. After the dbspace backup tapes have been restored, Universal 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, Universal Server does not check the constraints until the transaction is committed. If a constraint error occurs while the transaction is being committed, Universal 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 crashes before the children rows are deleted.

- **Distributed transactions**

Each Universal Server 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 Universal Server database servers that is performing the transaction. For more information, see [“Two-Phase Commit and Logical-Log Records” on page 34-30](#).

- **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 Universal Server data replication uses logical-log records, see [“How Does Data Replication Work?” on page 29-8](#).

What Database Server Activity Is Logged?

Universal Server does not generate log records for every operation because it does not need a record of every action. Universal Server needs log records only to perform the functions listed under [“Which Database Server Processes Require Logging?” on page 20-3](#). Also, the space required to store a record of everything the database server did would quickly be overwhelming.

The logical-log records themselves are of variable length. This arrangement increases the number of log records that can be written to a page in the logical-log buffer. However, Universal Server often flushes the logical-log buffer before the page is full.

Two types of logged activity are possible in Universal 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, refer to [Chapter 41, “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 20-9.) 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 Universal 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
- Sbspace metadata

For more information, see [“Are Smart Large Objects Logged?”](#) on page 20-8.

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 Simple Large Objects Logged?

Simple-large-object data is potentially too voluminous to include in a logical-log record. If simple-large-object data were always included, the many kilobytes of data per simple large object could overwhelm the space allocated for the logical log. However, not all simple large objects are that large, and not every simple large object would overwhelm the logical log.

Universal Server assumes that you designed your databases so that smaller simple large objects are stored in dbspaces, and larger simple large objects are stored in blobspaces. (For a discussion of locating simple-large-object data, see the [INFORMIX-Universal Server Performance Guide](#).) Based on this assumption, Universal Server takes the following action:

- Universal Server includes simple large object data in log records for simple large objects stored in dbspaces.
- Universal Server does not include simple large object data in log records for simple large objects stored in blobspaces.

Universal Server still needs access to the simple large object data for blobspace simple large objects in order to fulfill the goals of logging, explained in [“Which Database Server Processes Require Logging?” on page 20-3](#). 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 22-28](#) describes this mechanism.

Are Smart Large Objects Logged?

Sbspaces, as described in [“What Is an Sbspace?” on page 14-24](#), contain two components: metadata and user data. Metadata is *always* logged, regardless of the logging setting for the database, and even if logging is disabled for the database. User data is logged only at the request of a database administrator.

Metadata describes critical characteristics of smart large objects stored in a particular sbspace. If the metadata were to be damaged or become inaccessible, the sbspace would be corrupted and the smart large objects within that sbspace would be unrecoverable. Therefore, even if logging is turned off for a database, sbspace metadata is always logged.

Logging sbspace metadata ensures that the metadata can always be recovered to a transaction consistent state. Logging sbspace metadata ensures sbspace integrity.

Smart-large-object user data, that is, data created by user applications and stored in an sbspace, can be optionally logged. However, care must be taken to ensure that the logical logs are large enough to hold data generated by user data logging.

Applications can elect to turn off user data logging when a smart large object is initially written and then re-enable user data logging to capture edits to the smart large object.

When Logging Begins

When you turn on logging for a database, Universal Server does not begin logging until you perform a level-0 backup. However, when you turn on logging for a smart large object, Universal Server begins logging changes to it immediately.



Warning: When you turn logging on for a smart large object you must immediately perform a level-0 backup to ensure that the object is fully recoverable.

If you simply turn on logging for a smart large object and do not perform a level-0 backup, Universal Server only logs changes to the smart large object from that time forward. You must perform a level-0 backup initially to capture the initial content of the smart large object.

If you turn on logging for a smart large object and do not perform a level-0 backup, when a failure occurs fast recovery will restore changes up to the point of the last checkpoint, if a checkpoint occurred after turning on logging. For more information on checkpoints and fast recovery, refer to [Chapter 12, “Shared Memory,”](#) and [Chapter 26, “What Is Fast Recovery?”](#)

If you use ON-Bar for your backup and restore tool, refer to the [INFORMIX-Universal Server Backup and Restore Guide](#) for more information on performing backups. If you use ON-Archive or **ontape** for your backup and restore tool, refer to the [INFORMIX-Universal Server Archive and Backup Guide](#).

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 changing the database-logging status, refer to [“Who Can Set or Change Logging Status” on page 20-13](#) and to [Chapter 21, “Managing Database-Logging Status.”](#)

Even if you turn off transaction logging for all databases in an Universal Server database server, Universal Server always logs some events, as listed in [“Activity That Is Always Logged” on page 20-6](#).

The Database-Logging Status

Every database that Universal 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 Universal Server utilities, as explained in [“Monitoring Databases” on page 33-47](#). 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 12-6](#), information that Universal Server manages passes through shared memory to disk. Logical-log records are no exception. Before Universal 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 12-52](#).

In one sense, all Universal Server logging is buffered because all log records pass through the logical-log buffer in shared memory before Universal Server writes them to the logical log on disk. However, the point at which Universal 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. Universal 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 Universal 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 Universal Server 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, Universal 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 20-3](#).

If you are satisfied with your recovery source, you can decide not to use transaction logging for a database to reduce the amount of Universal 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, Universal 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 Universal 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 Universal Server administrator can change the logging status. This topic is described in [Chapter 21, “Managing Database-Logging Status.”](#) Ordinary end users cannot change the 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](#).

Managing Database-Logging Status

About Changing Logging Status	21-3
Modifying Database-Logging Status with ON-Archive	21-5
Turning On Transaction Logging with ON-Archive	21-5
Canceling a Logging Operation with ON-Archive	21-6
Ending Logging with ON-Archive	21-6
Changing Buffering Status with ON-Archive	21-6
Making a Database ANSI Compliant with ON-Archive	21-7
Modifying Database-Logging Status with ontape	21-7
Turning On Transaction Logging with ontape	21-7
Ending Logging with ontape	21-8
Changing Buffering Status with ontape	21-8
Making a Database ANSI Compliant with ontape	21-8
Modifying Database Logging Status with ON-Monitor	21-9

T

his chapter provides instructions on changing the database-logging status for databases that INFORMIX-Universal Server manages. As a Universal 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, refer to [Chapter 20, “What Is Logging?”](#)

To find out the current logging status of a database, see [“Monitoring Databases” on page 33-47](#).

About Changing Logging Status

[Figure 21-1 on page 21-4](#) shows which database-logging-status transitions the Universal Server administrator can perform and 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 a level-0 dbspace backup is performed on all the dbspaces and blobspaces that contain data in the database. Use the same backup tool (ON-Archive or **ontape**) to add logging and create the dbspace backup.

To change the logging status of ANSI-compliant databases, you must unload and reload the data.

Figure 21-1
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:

- To make any change in the logging status of a database, no users can access the database. Once you start to make the change, Universal 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, Universal Server must be in on-line mode in order to make changes. For **ontape** or ON-Monitor, Universal Server can be in either on-line or quiescent mode.

- If a failure occurs during a logging-mode change, check the flags for the database (see [“Monitoring Databases” on page 33-47](#)) after you restore the database server (or dbspace).
- Once you choose either buffered or unbuffered logging, you can change from either logging status to the other within 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 appears in the [INFORMIX-Universal Server Archive and Backup Guide](#).

Turning On Transaction Logging with ON-Archive

Before you make this change, read [“About Changing Logging Status” on page 21-3](#).

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 21-3](#).

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 21-3](#).

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 21-3](#). 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 the [INFORMIX-Universal Server Archive and Backup Guide](#).

Turning On Transaction Logging with ontape

Before you make this change, read [“About Changing Logging Status” on page 21-3](#).

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 21-3](#).

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 21-3](#).

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 21-3](#). Once you change the logging status to ANSI compliant, you cannot easily change it again. You must unload and reload the data. Techniques for unloading and loading data are discussed in 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.

Modifying Database Logging Status with ON-Monitor

Before you make any changes, read [“About Changing Logging Status” on page 21-3](#).

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.

What Is the Logical Log?

What Is the Logical Log?	22-3
What Is a Logical-Log File?	22-4
How Big Should the Logical Log Be?	22-5
Performance Considerations	22-5
Long-Transaction Consideration	22-6
Smart-Large-Object User-Data Logging Consideration	22-6
Smart-Large-Object Long-Transaction Consideration	22-7
Logical-Log Size Guidelines	22-7
Determining the Size of the Logical Log	22-8
Preserving Log Space for ON-Archive Tasks	22-8
Enabling the Logs-Full High-Water Mark	22-10
Emergency Log Backup	22-10
Building the System-Monitoring Interface	22-10
Recovery	22-10
Small Logs, Many Users	22-11
Administrative Activity When Logs Need Backing Up	22-11
What Should Be the Size and Number of Logical-Log Files?	22-12
Where Should Logical-Log Files Be Located?	22-13
How Are Logical-Log Files Identified?	22-13
What Are the Status Flags of Logical-Log Files?	22-14
Point-In-Time Recovery	22-16
Why Do Logical-Log Files Need to Be Backed Up?	22-16

When Are Logical-Log Files Freed?	22-17
When Does Universal Server Attempt to Free a Log File?	22-17
What Happens If the Next Logical-Log File Is Not Free?	22-17
Avoiding Long Transactions	22-19
Factors That Influence the Rate at Which Logical-Log Files	
Fill	22-19
Factors That Prevent Closure of Transactions	22-21
Setting High-Water Marks	22-21
What Are the Logical-Log Administration Tasks Required for	
Blobspaces?	22-23
Switching Logical-Log Files to Activate Blobspaces	22-23
Switching Logical-Log Files to Activate New Blobspace Chunks	22-23
Backing Up Logical-Log Files to Free Blobpages	22-24
Why Do You Have to Back Up Logical-Log Files to Free	
Blobpages?	22-24
What Is the Logging Process?	22-26
Dbospace Logging	22-26
Read Page into Shared-Memory Buffer Pool	22-27
Copy the Page Buffer into the Physical-Log Buffer	22-27
Read Data into Buffer and Create Logical-Log Record	22-27
Flush Physical-Log Buffer to the Physical Log	22-27
Flush Page Buffer	22-28
Flush Logical-Log Buffer	22-28
Blobspace Logging.	22-28

As an INFORMIX-Universal Server administrator, you have responsibilities for configuring and running 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 Universal Server logging process.

For more information on backing up logical-log files, a critical part of managing logical-log files, refer to the [*INFORMIX-Universal Server Archive and Backup Guide*](#).

For information on how to perform other logical-log tasks, refer to [Chapter 23, “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, Universal Server generates and stores log records. Universal 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 Universal Server backup tapes plus Universal Server logical-log files contains a complete copy of your Universal Server data.

All the databases managed by a single Universal 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 20, “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 20-13](#).

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 Universal 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 Universal 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, Universal Server can suspend processing and, in the worst case, shut down.

The Universal 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 22-12](#)
- [“Where Should Logical-Log Files Be Located?” on page 22-13](#)

The Universal 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 22-16](#)
- [“When Are Logical-Log Files Freed?” on page 22-17](#)

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 22-23](#).

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 up, you have to back them up. (See [“Why Do Logical-Log Files Need to Be Backed Up?” on page 22-16](#).) The backup process can hinder transaction processing 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, Universal Server must write a new checkpoint record to the current logical-log file. (See [“When Are Logical-Log Files Freed?” on page 22-17.](#)) 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 22-19.](#)

Smart-Large-Object User-Data Logging Consideration

If you choose to log smart-large-object user data, you should be aware that this option can potentially generate large amounts of logical log traffic, possibly filling the logical logs. One way to reduce the volume of logical log entries for fully logged smart large objects is to turn off logging when the smart large object is originally loaded and then re-enable logging after the loading is complete.

Smart-Large-Object Long-Transaction Consideration

Smart large objects can be used in situations where the data collection process for a single smart large object lasts for long periods of time. Consider, for example, an application recording many hours of low-quality audio information. While the amount of data collected may be modest, the recording session may be quite long, resulting in a long transaction condition. For more information on the long-transaction situation, refer to [“Avoiding Long Transactions” on page 22-19](#).

Logical-Log Size Guidelines

You use the LOGSIZE parameter to set the size of the logical log. It is difficult to predict how much logical-log space your Universal Server system requires until it is fully in use. The following expression gives the *minimum* total-log-space configuration, in kilobytes, that Informix recommends:

$$\text{LOGSIZE} = (\text{users} * \text{maxrows}) * 512$$

Set *users* to the maximum number of users that you expect to access Universal Server concurrently. If you set the NETTYPE parameter, you can use the value 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.

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 23-4](#). 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 on this topic, see the [INFORMIX-Universal Server Performance Guide](#).

Determining the Size of the Logical Log

The LOGFILES parameter gives the number of logical-log files. The LOGSIZE parameter gives the size of the logical-log files created when Universal Server initializes disk space. The Universal Server administrator sets both of these configuration parameters in the ONCONFIG file. 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
```

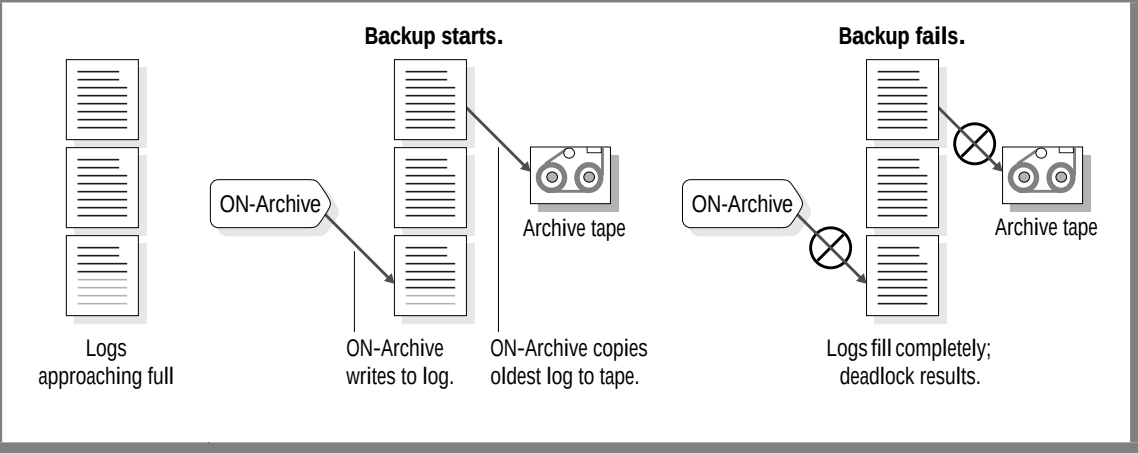
If you add logical-log files that are not the size specified by LOGSIZE, you cannot use the (LOGFILES * LOGSIZE) expression to calculate the size of the logical log. Instead, you need to add the sizes for each individual log file on disk. For information on how to access the size of logical-log files, refer to [“Monitoring Logical-Log Files” on page 33-48](#).

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.

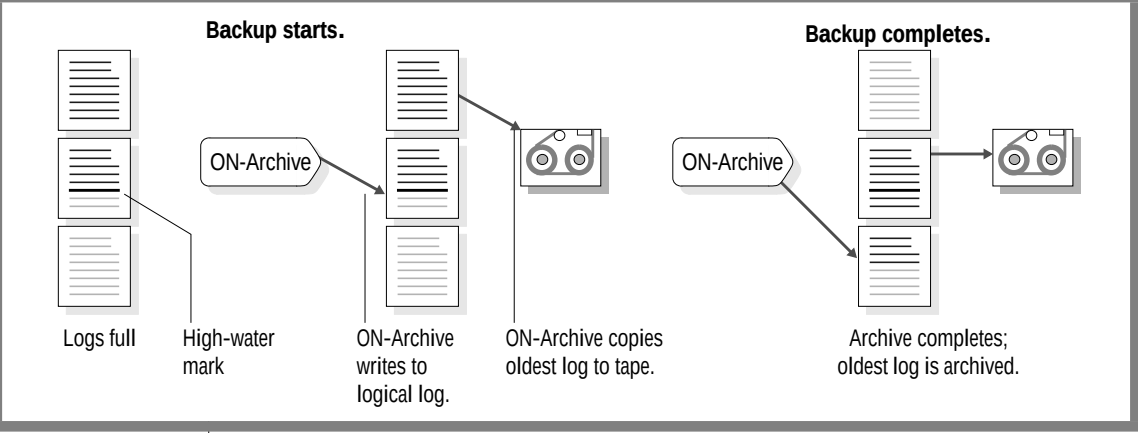
Prior to marking a backed-up log as free, Universal Server updates the ON-Archive catalog tables to record the occurrence of the logical-log backup. This update itself generates logical-log activity, leading to a possible deadlock as the logs continue to fill. When a deadlock of this type occurs, the Universal Server administrator must use the emergency log backup procedure even though a continuous logical-log backup is already in progress. [Figure 22-1 on page 22-9](#) illustrates how a deadlock of this type occurs.

Figure 22-1
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, Universal Server blocks OLTP activity when the next-to-last log fills, rather than the last log, as it usually does. In doing so, Universal Server preserves the last logical-log file for record logging generated by administrative activities such as a backup of the logical log. Figure 22-2 illustrates how the high-water mark feature prevents logical-log deadlocks.

Figure 22-2
ON-Archive with High-Water Mark On



For more information on the emergency backup procedure, see the [*INFORMIX-Universal Server 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, Universal 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 Universal 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 Universal 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:

$$\text{Logical Log Size} < 2 * \text{page\_size} * \text{number of users}$$

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.
- If you plan to include many smart-large-objects in your database, you should be aware that smart-large-object metadata is *always* logged and increase your logical log size accordingly.
- If you plan to log smart-large-object user data, you must ensure that the log size is *considerably* larger than the amount of data being written.

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 blob space blobpages can be reclaimed. See [“Backing Up Logical-Log Files to Free Blobpages” on page 22-24](#).
- The number of logical-log files cannot exceed the value of the ONCONFIG parameter LOGSMAX.

Where Should Logical-Log Files Be Located?

When Universal 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 23-8](#).

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 Universal Server disk space. When the current logical-log file becomes full, Universal 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 22-16](#)), Universal Server reuses the disk space for the logical-log files. However, Universal Server continues to increment the unique ID numbers by one. [Figure 22-3 on page 22-14](#) illustrates the relationship between the logid numbers and the unique ID numbers.

Figure 22-3
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 33-48.

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:

- 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 Universal Server for recovery (rollback of a transaction or finding the last checkpoint record).

Backed-Up (B)	A log file has a <i>backed-up</i> status after the log file has been backed up.
Current (C)	A log file has a <i>current</i> status if Universal Server is currently filling the log file.
Last (L)	A log file has a status of <i>last</i> if the log file contains the most recent checkpoint record in the logical log. This file and subsequent files cannot be freed until Universal Server writes a new checkpoint record to a different logical-log file.

Figure 22-4 shows the possible log-status flag combinations.

Figure 22-4
Logical-Log Status Flags

Status Flag	Status of Logical-Log File
A-----	Log has been added since 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. Contains 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 33-48](#).

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. Using the UNTIL option of the **ondatartr** utility of ON-Archive, the entire system must be restored to a previous point in time. For additional information about restoring logical logs to a particular point in time, see the [INFORMIX-Universal Server Archive and Backup Guide](#).

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.

To perform a logical-log-file backup, use either ON-Archive or **ontape** depending on which of these utilities you use to create and maintain your dbspace and logical-log backups.

Logical-log-file backups can be initiated implicitly as part of continuous logging, or explicitly by the Universal Server administrator or operator, either through ON-Archive, or by executing **ontape**. For more information, see the [INFORMIX-Universal Server 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 Universal 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 Universal Server Attempt to Free a Log File?

Universal Server attempts to free logical-log files under the following conditions:

- When Universal Server first writes to a new logical-log file, it attempts to free the previous log.
- Each time that Universal 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 Universal Server attempts to switch to the next logical-log file but finds that the next log file in sequence is still in use, Universal Server immediately suspends all processing. Even if other logical-log files are free, Universal 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.

What Happens If the Next Logical-Log File Is Not Free?

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. However, 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 the [INFORMIX-Universal Server Archive and Backup Guide](#).

- The file contains an open transaction.

The open transaction is the long transaction discussed in “[Avoiding Long Transactions](#)” on page 22-19. In this situation, you have to recover Universal Server data from dbspace backup tapes in a full-system restore.

A situation whereby Universal Server must suspend processing because the next log file contains the last checkpoint never occurs. Universal 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, Universal 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 Universal 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 Universal 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 Universal 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 Universal 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 23-4](#).
- Number of logical-log records
The more logical-log records written to the logical log, the faster it fills. If databases managed by your Universal Server 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 20-10](#), 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 Universal 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.
- **Number of smart large objects**
Because smart-large-object metadata is always logged, a database with a great number of frequently updated smart large objects fills logical logs rapidly.
- **Number of smart large objects with user data logging enabled**
Smart large objects that have user data logging enabled and have a large volume of user data updates can fill logical logs at a *prodigious* rate.

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 Universal Server to complete the transaction. Repeated writes to the logical-log file increase the amount of CPU time that Universal 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

Universal 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 37-42](#). When the logical log reaches the long-transaction high-water mark, Universal 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, Universal 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 [“LTXEHW” on page 37-41](#). When the logical log reaches the exclusive-access, long-transaction high-water mark, Universal 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, Universal 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.

You must choose appropriate values for LTXHWM and LTXEHW so that Universal Server does not come off-line due to a long transaction. If you have recurring problems with long transactions using the default values of 80 for LTXHWM and 90 for LTXEHW, Informix recommends setting LTXHWM to 50 and LTXEHW to 60. If you follow this advice, however, you must also double the size of your logical logs to accommodate the lower settings.

The default values for the configuration parameters LTXHWM and LTXEHW have changed from 80 and 90 to 50 and 60, respectively. This change eliminates any risk of a long transaction having too little logspace in which to roll back. Universal Server initialization will emit 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. Universal 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 23-15](#).

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. Universal 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 23-15](#).

Backing Up Logical-Log Files to Free Blobpages

When you delete data stored in blobpage pages, those pages are not necessarily freed for reuse. The blobpage pages are only free when *both* of the following actions have occurred:

- The simple large object has been deleted, either through an UPDATE to the simple-large-object column on the row or by deleting the row.
- The logical log with the INSERT of the row that contains the simple large object 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 a simple large object, Universal Server does not write the actual simple-large-object data to the logical-log file. Universal Server writes only the blobpage 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 blobpage chunk. Each entry contains the following information:

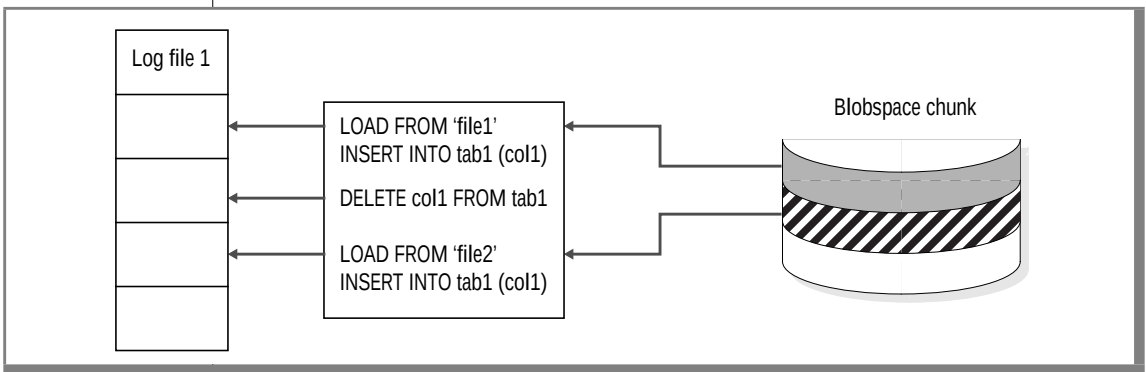
- A flag that indicates whether the page is used or free
- The unique ID of the logical-log file that was current when the page was written to
- The associated tblspace number of the data stored on the page

For more information on the free-map page, see [“Blobpage Page Types” on page 42-67](#).

When an application deletes a row that contains a simple large object, Universal Server marks each blobpage that is part of the deleted simple large object as FREE in the free-map overhead page. Universal Server writes a log record to the logical log to record the changes to the free-map page. If Universal Server has not yet backed up the logical-log file that contains the transaction that originally inserted the simple large object, Universal Server does not write over the deleted blobpages on subsequent simple large object inserts. Figure 22-5 illustrates this scenario.

Figure 22-5

No Write to Deleted Blobpages If Logical-Log File is Not Backed Up

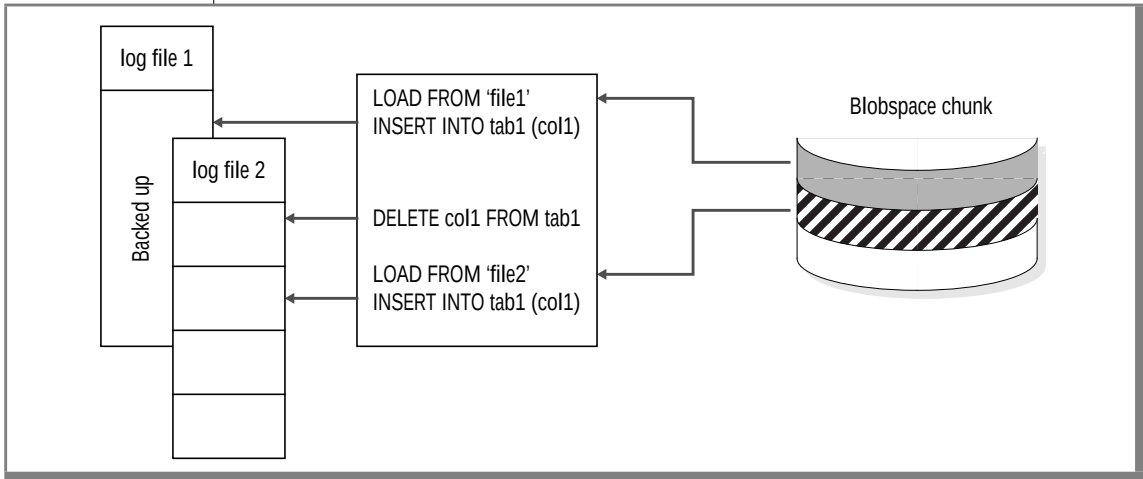


Universal Server does not write over the deleted blobpages in the scenario shown in Figure 22-5 because it might need to perform a fast recovery and roll forward the transaction that contains the original insert. If the original data was overwritten, Universal Server would have no way of reproducing the transaction.

When you back up the logical-log file that contains the original insert, however, Universal Server copies the actual simple-large-object data to tape. After the simple-large-object data is saved to tape and available for a recovery, Universal Server can write over the deleted blobpages, as shown in [Figure 22-6 on page 22-26](#).

Figure 22-6

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 blobpage logging. This information is not required for performing normal Universal Server administration tasks.

Dbospace Logging

Universal 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. Universal Server identifies the page on which the row resides and attempts to locate the page in the Universal Server shared-memory buffer pool. If the page is not already in shared memory, Universal Server reads the page from disk. [“How a Database Server Thread Accesses a Buffer Page” on page 12-42](#) explains this process in more detail.

Copy the Page Buffer into the Physical-Log Buffer

Before Universal Server modifies a dbospace data page, it stores a copy of the unchanged page in the physical-log page buffer. Universal 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 Universal 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 12-46](#).

Universal 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 Universal 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 12-52](#).

Flush Physical-Log Buffer to the Physical Log

Universal 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 12-46](#).

Flush Page Buffer

At some point after Universal Server flushes the physical-log buffer, Universal 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. Universal 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 12-45.](#)

Flush Logical-Log Buffer

To flush the logical-log buffer, Universal 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 12-52.](#)

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

Universal Server logs blobspace data, but the data does not pass through either shared memory or the logical-log files on disk. Universal 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 simple large objects are deleted from blobpages), but not the actual simple-large-object data. Blobspace simple-large-object 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 simple large object requires more than one blobpage, Universal 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 simple-large-object data is not included in the record (but the information about where the simple-large-object data is placed is included by way of the overhead pages).
3. When a logical-log backup begins, Universal Server uses the logical-log ID number stored in the blobspace free-map page to determine which blobpages to copy to tape. For more details on logical-log backups, see the [*INFORMIX-Universal Server Archive and Backup Guide*](#).

Managing Logical-Log Files

Adding a Logical-Log File	23-4
Using ON-Monitor to Add a Log File	23-4
Using onparams to Add a Log File	23-5
Adding a Log File with a New Size	23-5
Dropping a Logical-Log File.	23-6
Using ON-Monitor to Drop a Logical-Log File	23-7
Using onparams to Drop a Logical-Log File	23-7
Moving a Logical-Log File to Another Dbspace	23-8
An Example of Moving Logical-Log Files.	23-8
Changing the Size of Logical-Log Files	23-9
Changing Logical-Log Configuration Parameters	23-10
Changing LOGSIZE or LOGFILES	23-10
Using ON-Monitor to Change LOGSIZE or LOGFILES.	23-11
Using a Text Editor to Change LOGSIZE or LOGFILES.	23-11
Changing LOGSMAX, LTXHWM, or LTXEHWL	23-12
Changing LOGSMAX, LTXHWM, or LTXEHWL with ON-Monitor	23-12
Editing the ONCONFIG File to Change LOGSMAX, LTXHWM, or LTXEHWL	23-13
Freeing a Logical-Log File	23-13
Freeing a Log File with Status A	23-13
Freeing a Log File with Status U	23-14
Freeing a Log File with Status U-B	23-14
Freeing a Log File with Status U-C or U-C-L.	23-14
Freeing a Log File with Status U-B-L	23-15
Switching to the Next Logical-Log File	23-15

T

his chapter contains information on managing the INFORMIX-Universal Server logical-log files. 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

You must manage logical-log files even if none of your databases use transaction logging.

For background information regarding the logical log, refer to [Chapter 22, “What Is the Logical Log?”](#)

For instructions on backing up logical-log files with ON-Bar, refer to the [INFORMIX-Universal Server Backup and Restore Guide](#). For instructions on backing up logical-log files with ON-Archive or **ontape**, refer to the [INFORMIX-Universal Server 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 an dbspace backup (quiescent or on-line), and Universal 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 the backup tool you usually use to create the level-0 dbspace backup.

You can use either ON-Monitor or **onparams** to add the log file. You must use the **onparams** utility to add a new log file with a different size than LOGSIZE. If you use ON-Monitor, the size is always the value specified by 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 LTXEHW” on page 23-12](#)) and reinitialize shared memory for the change to take effect.

You must be logged in as either **informix** or **root** to make this change.

Using ON-Monitor to Add a Log File

Bring Universal Server to quiescent mode. Select the Parameters menu, 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 [INFORMIX-Universal Server Backup and Restore Guide](#). For information on creating a level-0 dbspace backup with ON-Archive or **ontape**, refer to the [INFORMIX-Universal Server 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 the [INFORMIX-Universal Server Backup and Restore Guide](#). For information on creating a level-0 dbspace backup with ON-Archive or **ontape**, refer to the [INFORMIX-Universal Server Archive and Backup Guide](#).

For reference information on using **onparams** to add a logical-log file, see “Add a Logical-Log File” on page 39-51.

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 the [INFORMIX-Universal Server Backup and Restore Guide](#). For information on creating a level-0 dbspace backup with ON-Archive or **ontape**, refer to the [INFORMIX-Universal Server Archive and Backup Guide](#).

For reference information on using **onparams** to add a logical-log file, see [“Add a Logical-Log File” on page 39-51](#).

Dropping a Logical-Log File

You can drop a log to increase the amount of the disk space available within a dbspace.

Universal 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 Universal 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 33-48](#).

To make this change, you must log in as either **informix** or **root**, and Universal Server must be in quiescent mode.

Using ON-Monitor to Drop a Logical-Log File

Select the Parameters menu, Drop-Log option 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 the [INFORMIX-Universal Server Backup and Restore Guide](#). For information on creating a level-0 dbspace backup with ON-Archive or **ontape**, refer to the [INFORMIX-Universal Server 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 the [INFORMIX-Universal Server Backup and Restore Guide](#). For information on creating a level-0 dbspace backup with ON-Archive or **ontape**, refer to the [INFORMIX-Universal Server Archive and Backup Guide](#).

See “Drop a Logical-Log File” on page 39-52 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 22-13](#). To find out the location of logical-log files, see [“Monitoring Logical-Log Files” on page 33-48](#).

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.”](#)

An Example of Moving Logical-Log Files

Universal 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 23-13](#).
2. Verify that the value of LOGSMAX is greater than or equal to the number of log files after the move plus three. 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 LTXEHW” on page 23-12](#).
3. Drop all but three of the logical-log files. See [“Dropping a Logical-Log File” on page 23-6](#).

You cannot drop the current log file. If you have only three logical-log files in the root dbspace, skip this step.

4. Add the new log files to the different dbspace. See [“Adding a Logical-Log File” on page 23-4](#).

In this case, add six new log files to **dbspace_1**.

5. Create a level-0 dbspace backup of the root dbspace to make the new log files available to Universal Server. For information on creating a level-0 dbspace backup with ON-Bar, refer to the [INFORMIX-Universal Server Backup and Restore Guide](#). For information on creating a level-0 dbspace backup with ON-Archive or **ontape**, refer to the [INFORMIX-Universal Server 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 23-15](#).
7. Back up the former *current log file* to free it. For information on backing up logical-log files with ON-Bar, refer to the [INFORMIX-Universal Server Backup and Restore Guide](#). For information on backing up logical-log files with ON-Archive or **ontape**, refer to the [INFORMIX-Universal Server 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 23-6](#).

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 23-5](#).
- 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 23-10](#).

Changing Logical-Log Configuration Parameters

The following configuration parameters affect the logical-log file and how Universal Server works with it:

- LOGSIZE (described on [page 37-35](#))
- LOGFILES (described on [page 37-34](#))
- LOGSMAX (described on [page 37-36](#))
- LTXHWM (described on [page 37-42](#))
- LTXEHWM (described on [page 37-41](#))

The following sections explain the procedure for changing each of these parameters.

Changing LOGSIZE or LOGFILES

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 23-4](#). 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.

Using ON-Monitor to Change LOGSIZE or LOGFILES

You can use ON-Monitor to change the value of LOGSIZE or LOGFILES.

Unload all Universal Server data. For information about unloading data, refer to the [Informix Migration Guide](#). You cannot rely on dbspace backup tapes to unload and restore the data because a restore returns the parameters to their previous value.

Select the Parameters menu, Initialize option to reinitialize disk space. 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**. Proceed with Universal Server disk-space initialization.

After Universal Server disk space is reinitialized, re-create all databases and tables. Then reload all Universal Server data. For information about the **onload** utility, 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.

Change the value of LOGSIZE or LOGFILES.

Unload all Universal Server data. For information about moving tables and data, refer to the [Informix Migration Guide](#). You cannot rely on dbspace backup tapes to unload and restore the data because a restore returns the parameters to their previous value.

Use **oninit** to reinitialize disk space. After Universal Server disk space is reinitialized, re-create all databases and tables. Then reload all Universal Server data. For information about the **onload** utility, see the [Informix Migration Guide](#).

Changing LOGSMAX, LTXHWM, or LTXEHWM

You can change the value of LOGSMAX, LTXHWM, or LTXEHWM with the following methods:

- 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.

Changing LOGSMAX, LTXHWM, or LTXEHWM with ON-Monitor

You can use ON-Monitor to change LOGSMAX, LTXHWM, or LTXEHWM while Universal Server is in on-line mode.

Select the Parameters menu, 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 the value of:	Change the 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 13-15.](#)

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 13-15](#).

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 22-14](#).

You might want to free a logical-log file for the following reasons:

- So that Universal 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 33-48](#).

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 the [INFORMIX-Universal Server Backup and Restore Guide](#). For information on creating a level-0 dbspace backup with ON-Archive or **ontape**, refer to the [INFORMIX-Universal Server 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 the [INFORMIX-Universal Server Backup and Restore Guide](#). For information on backing up logical-log files with ON-Archive or **ontape**, refer to the [INFORMIX-Universal Server 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” in the following section or “[Freeing a Log File with Status U-B-L](#)” on page 23-15.

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 Universal Server to quiescent mode. See “[Immediately from On-Line to Quiescent](#)” on page 8-6. 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. Now 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 the [INFORMIX-Universal Server Backup and Restore Guide](#). For information on backing up logical-log files with ON-Archive or **ontape**, refer to the [INFORMIX-Universal Server 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 23-14](#) or “Freeing a Log File with Status U-B-L” in the following section.

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, Universal Server must perform a new checkpoint. To force a checkpoint, either select the ON-Monitor Force-Ckpt option or execute the following command:

```
% onmode -c
```

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

Universal 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.)

What Is Physical Logging?

What Is Physical Logging?	24-3
What Is the Purpose of Physical Logging?	24-4
Fast Recovery Uses Physically Logged Pages	24-4
Backup Uses Physically Logged Pages	24-4
What Database Server Activity Is Physically Logged?	24-4
Are Blobs Physically Logged?	24-5
How Big Should the Physical Log Be?	24-5
Can the Physical Log Become Full?	24-7
Where Is the Physical Log Located?	24-8
Details of Physical Logging	24-9
Page Is Read into the Shared-Memory Buffer Pool.	24-9
A Copy of the Page Buffer Is Stored in the Physical-Log Buffer	24-10
Change Is Reflected in the Data Buffer.	24-10
Physical-Log Buffer Is Flushed to the Physical Log	24-10
Page Buffer Is Flushed	24-10
When a Checkpoint Occurs	24-11
How the Physical Log Is Emptied	24-11

T

his chapter defines the terms and explains the concepts that you need to know to perform effectively the tasks described in [Chapter 25, “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 its size and location
- Details of the physical-logging process

What Is Physical Logging?

Physical logging is the process of storing the pages that the INFORMIX-Universal Server is going to change before the changed pages are actually recorded. Before Universal 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 Universal Server stores before-images.

Universal 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 12-56.](#)) Once a checkpoint occurs, Universal Server empties the physical log (except in the special circumstances explained in [“Can the Physical Log Become Full?” on page 24-7.](#))

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 Universal Server data. These snapshots are potentially used in two activities: fast recovery and Universal Server backup.

Fast Recovery Uses Physically Logged Pages

After a failure, Universal 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, Universal Server can return all data to physical and logical consistency, up to the point of the most-recently completed transaction. [Chapter 26, “What Is Fast Recovery?”](#) explains this procedure in more detail.

Backup Uses Physically Logged Pages

When you perform an on-line dbspace backup, Universal Server checks disk pages to see which should be backed up. As part of this process, Universal Server periodically reads the pages in the physical log. If Universal Server finds pages in the physical log that meet the backup criterion, they are copied to the backup tape. For details, see the [INFORMIX-Universal Server Archive and Backup Guide](#).

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 Universal Server address. This situation usually occurs when the page was used by some other Universal Server database server or a table that was dropped.
- Pages that Universal 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 Universal Server stores the before-images in the physical log only until the next checkpoint. To control the amount of data that Universal Server logs, you can tune the checkpoint interval configuration parameter CKPTINTVL.

Are Blobs Physically Logged?

The Universal Server pages in the physical log can be any Universal Server page except a blob space blob page. Even overhead pages (such as chunk free-list pages, blob space free-map pages, and blob space 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 blob space blob pages are not. For further information about blob space logging, see [“Are Simple Large Objects Logged?” on page 20-7](#).

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:

$$\text{Physical Log Size} = \text{userthreads} * \text{max\_log\_pages\_per\_critical\_section} * 4$$

This formula is based on how much physical logging space Universal 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 12-55](#)) and perform physical logging of the maximum number of pages in their critical section, Universal 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 displays the number of maximum concurrent user threads. Substitute this number for *userthreads* in the formula.

The maximum number of pages (*max_log_pages_per_critical_section*) that Universal 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 Universal Server perform?

Operations that do not perform updates do not generate before-images. If the applications that use your database server do not do 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.

Universal 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, that 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 37-57.)

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 Universal 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 12-55](#).) 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 Universal Server processes tblspace blobs stored in a database created with transaction logging, each portion of the blob that Universal 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, Universal 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, Universal Server sends the following message to the message log:

```
Physical log file overflow
```

Universal 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, LTXEHW, is reached. (See [“Avoiding Long Transactions” on page 22-19.](#)) After the LTXEHW 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 LTXEHW was reached.

Where Is the Physical Log Located?

When Universal 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 37-57.](#)) 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 25-3.](#))

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.

Universal 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, Universal Server identifies the page on which the row resides and attempts to locate the page in the Universal Server shared-memory buffer pool. If the page is not already in shared memory, it is read into the resident portion of Universal 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. Universal Server reflects these changes in the shared-memory data buffer.

Data from the application is passed to Universal 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

Universal 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 12-55](#).

When a Checkpoint Occurs

A checkpoint can occur at any point in the physical-logging process. After a checkpoint occurs, Universal 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. Universal Server empties the physical log logically, allowing current entries to be overwritten.

How the Physical Log Is Emptied

Universal 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.

Managing the Physical Log

Changing the Physical-Log Location and Size	25-3
Why Change Physical-Log Location and Size?	25-3
Before You Make the Changes.	25-4
Using ON-Monitor to Change Physical-Log Location or Size . . .	25-4
Using a Text Editor to Change Physical-Log Location and Size . .	25-5
Using onparams to Change Physical-Log Location or Size	25-5

T

his chapter describes how to change the configuration parameters associated with the physical log. For background information about the physical log, see [Chapter 24, “What Is Physical Logging?”](#)

Changing the Physical-Log Location and Size

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** 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 INFORMIX-Universal Server recovery.

Why Change Physical-Log Location and Size?

You can move the physical-log file to try to improve performance. When Universal 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 24-8](#). For advice on sizing the physical log see [“How Big Should the Physical Log Be?” on page 24-5](#).

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 Universal 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 33-60](#).

To change the physical-log location or size with either ON-Monitor or **onparams**, you must log in as user **root** or **informix**.

Using ON-Monitor to Change Physical-Log Location or Size

Select the Parameters menu, Physical-Log option 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 Universal 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. Universal Server displays messages that Universal 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 Universal Server is in on-line mode:

- `PHYSFILE` [page 37-57](#)
- `PHYSDBS` [page 37-57](#)

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 39-50.

To change the size and location of the physical log, execute the following command after you bring Universal 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.

Using onparams to Change Physical-Log Location or Size

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 for your backup and restore tool, refer to the [INFORMIX-Universal Server Backup and Restore Guide](#) for information on pilfering a level-0 dbspace backup. For information on performing a level-0 dbspace backup using ON-Archive or **ontape**, refer to the [INFORMIX-Universal Server Archive and Backup Guide](#).

What Is Fast Recovery?

What Is Fast Recovery?	26-3
When Is Fast Recovery Needed?	26-4
When Does the Database Server Initiate Fast Recovery?	26-4
Fast Recovery and Buffered Logging	26-4
Fast Recovery and No Logging	26-5
Details of Fast Recovery	26-5
Returning to the Last-Checkpoint State	26-6
Finding the Checkpoint Record in the Logical Log.	26-6
Rolling Forward Logical-Log Records	26-7
Rolling Back Incomplete Transactions	26-7

T

his chapter describes the fast-recovery feature of INFORMIX-Universal Server. 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 Universal Server detects these failures
- The details of how fast recovery works

What Is Fast Recovery?

Fast recovery is an automatic, fault-tolerant feature that Universal 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 Universal Server went off-line, it did so in uncontrolled conditions. If so, fast recovery returns Universal Server to a state of physical and logical consistency, as described in [“Details of Fast Recovery” on page 26-5](#).

If the fast-recovery process finds that Universal Server came off-line in a controlled manner, the fast-recovery process terminates, and Universal Server moves to on-line mode.

When Is Fast Recovery Needed?

Fast recovery restores Universal Server to physical and logical consistency after any failure that results in the loss of the contents of memory for Universal 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:

- Universal Server is processing tasks for more than 40 users.
- Dozens of transactions are on-going.
- Without warning, the operating system fails.

How does Universal 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 Universal Server to quiescent mode or on-line mode from off-line mode, Universal Server checks if fast recovery is needed.

As part of shared-memory initialization, Universal Server checks the contents of the physical log. The physical log is empty when Universal Server shuts down under control. The move from on-line mode to quiescent mode includes a checkpoint, which flushes the physical log. Therefore, if Universal Server finds pages in the physical log, Universal 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 20-11](#)), 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 Universal 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:

- Universal Server uses the physical log to return to the most recent point of known *physical consistency*, the most recent checkpoint.
- Universal 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, Universal 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 Universal Server writes each before-image page in the physical log to shared memory and then back to disk, changes to Universal Server data since the time of the most recent checkpoint are undone. Figure 26-1 illustrates this step.

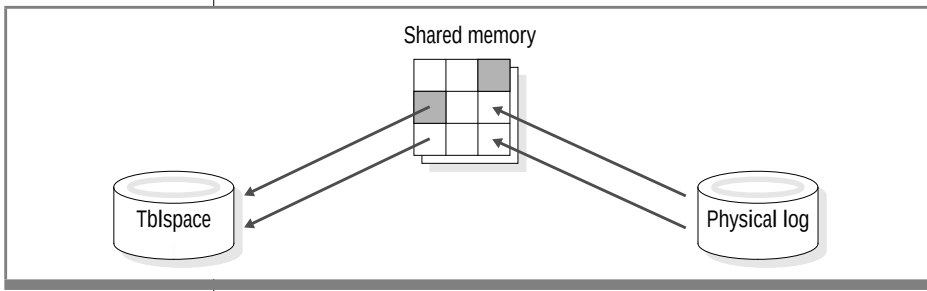


Figure 26-1
*Writing All
Remaining Before-
Images in the
Physical Log Back
to Disk*

Finding the Checkpoint Record in the Logical Log

In the second step, Universal 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. (See [“PAGE_CKPT” on page 42-8.](#))

The address information from PAGE_CKPT also identifies the location of all logical-log records written after the most recent checkpoint. Figure 26-2 illustrates this step.

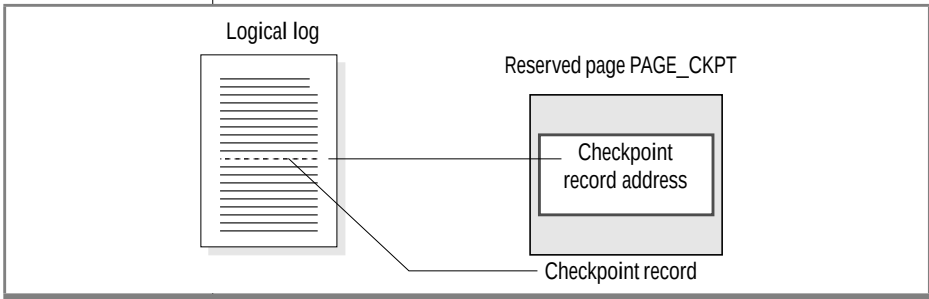


Figure 26-2
Locating the Most Recent Checkpoint Record in the Logical Log

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 26-3 illustrates this step.

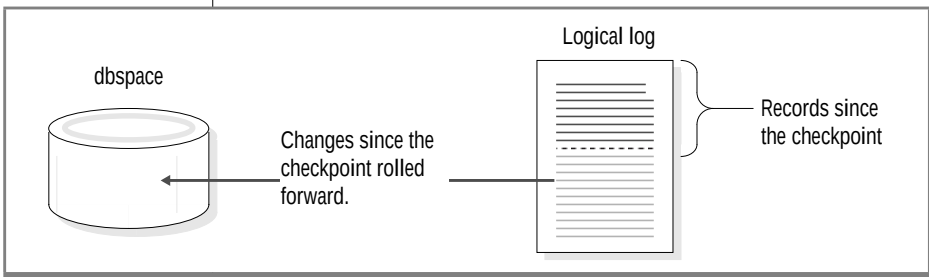


Figure 26-3
Rolling Forward the Logical-Log Records Written Since the Most Recent Checkpoint

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, refer to [“How the Two-Phase Commit Protocol Handles Failures”](#) on page 34-10.) This rollback procedure ensures that all databases are left in a consistent state.

Rolling Back Incomplete Transactions

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 Universal Server because a log file is not freed until all transactions contained within it are closed. Figure 26-4 illustrates the rollback procedure. When fast recovery is complete, Universal Server goes to quiescent or on-line mode

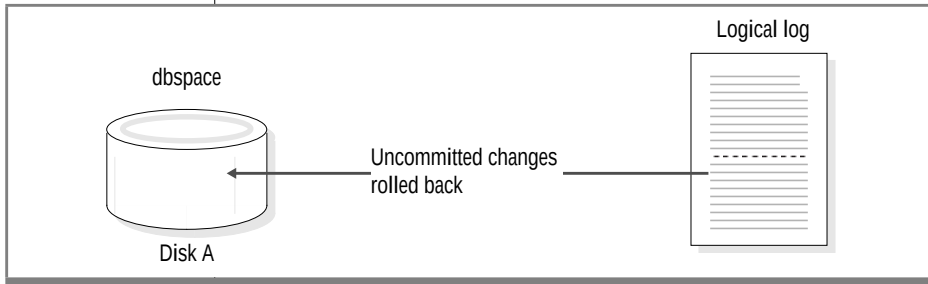


Figure 26-4
*Rolling Back All
Incomplete
Transactions*

Fault Tolerance

Section VI



What Is Mirroring?

What Is Mirroring?	27-4
What Are the Benefits of Mirroring?	27-4
What Are the Costs of Mirroring?	27-5
What Happens If You Do Not Mirror?	27-5
What Should You Mirror?	27-5
What Mirroring Alternatives Exist?	27-6
The Mirroring Process.	27-7
What Happens When You Create a Mirrored Chunk?	27-7
What Are Mirror Status Flags?	27-8
What Is Recovery?.	27-8
What Happens During Processing?	27-9
Disk Writes to Mirrored Chunks	27-9
Disk Reads from Mirrored Chunks.	27-9
Detecting Media Failures	27-10
Recovering a Chunk.	27-10
What Happens If You Stop Mirroring?.	27-11
What Is the Structure of a Mirrored Chunk?	27-11

T

he first part of this chapter answers the following basic questions about the INFORMIX-Universal 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 28, “Using Mirroring.”](#)

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 27-1. 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.

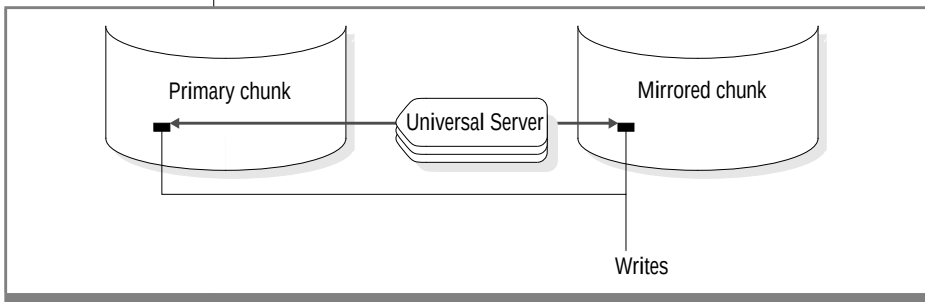


Figure 27-1
*Writing Data to Both
the Primary Chunk
and the Mirrored
Chunk*

Mirroring is not supported on disks that are managed over a network. The same Universal 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 Universal Server administrator with a means of recovering data without having to take Universal 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 Universal 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 Universal 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 Universal Server performs reads and writes for mirrored chunks, see [“What Happens During Processing?” on page 27-9](#).

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, Universal 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, Universal Server cannot access the data stored on that chunk. If the chunk contains logical-log files, the physical log, or the root dbspace, Universal Server goes off-line immediately. If the chunk does not contain logical-log files, the physical log, or the root dbspace, Universal 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, Universal 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 Universal 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 Universal 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 Universal Server strategy of performing writes to mirrored chunks in parallel (see [“Disk Writes to Mirrored Chunks” on page 27-9](#)) helps to reduce the time required for this step. In addition, Universal Server mirroring uses split reads to improve read performance. (See [“Disk Reads from Mirrored Chunks” on page 27-9](#).) Operating-system mirroring features that do not use parallel mirror writes and split reads might provide inferior performance.

Nothing prevents you from running Universal 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 Universal Server mirroring and the mirroring feature provided by your operating system. For example, you might have both Universal Server data and non-Universal Server data on a single disk drive. You could use the operating-system mirroring to mirror the non-Universal Server data and Universal Server mirroring to mirror the Universal 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 Universal 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 Universal 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 28, “Using Mirroring.”](#)

What Happens When You Create a Mirrored Chunk?

When you specify a mirrored chunk, Universal 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 Universal 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 Universal 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 Universal 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, Universal Server is unable to take advantage of the mirrored log files.

For similar reasons, you cannot mirror a dbospace that contains a logical-log file while a dbospace backup is being created. The new information that must appear in the first block of the dbospace backup tape cannot be copied there once the backup has begun.

For more information on creating mirrored chunks, refer to [Chapter 28, “Using Mirroring.”](#)

What Are Mirror Status Flags?

Dbspaces and blobspaces have status flags that indicate whether the dbospace or blobspace is mirrored, unmirrored, or mirrored but requiring a level-0 backup of the root dbospace 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 dbospace, or being recovered.

For descriptions of these chunk status flags, refer to [“-d Option” on page 39-84](#). For information on how to display these status flags, refer to [“Monitoring Chunk Status” on page 33-55](#).

What Is Recovery?

When Universal 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, Universal 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, Universal 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 28-11](#).

What Happens During Processing?

This section discusses some of the details of disk I/O for mirrored chunks and how Universal Server handles media failure for these chunks.

Disk Writes to Mirrored Chunks

During Universal Server processing, Universal 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 10-28](#).

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

Universal 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 27-2 on page 27-10](#) illustrates a split read.

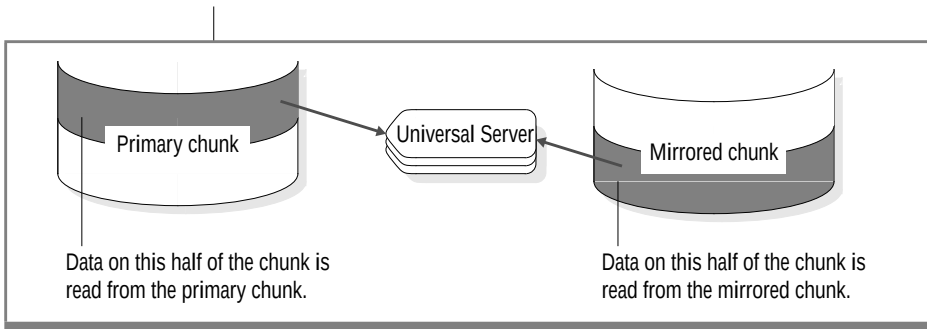


Figure 27-2
*Split Read Reducing
the Maximum
Distance Over
Which the Disk Head
Must Travel*

Detecting Media Failures

Universal Server checks the return code when it first opens a chunk and after any read or write. Whenever Universal 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 27-8](#).

If Universal 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

Universal 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, Universal 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 blob space chunks contain blob space overhead pages; mirrors of db space chunks contain db space overhead pages. For information on these structures, refer to [“Structure of a Mirrored Chunk” on page 42-16](#).

A display of disk-space use, provided by one of the methods discussed under [“Monitoring Chunks” on page 33-60](#), 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.

Using Mirroring

Steps Required for Mirroring Data	28-3
Enabling Mirroring.	28-4
Enabling Mirroring with ON-Monitor	28-5
Enabling Mirroring by Editing the ONCONFIG File.	28-5
Allocating Disk Space for Mirrored Data	28-5
Starting Mirroring	28-6
Mirroring the Root Dbspace During Initialization	28-7
Setting MIRRORPATH and MIRROROFFSET with ON-Monitor.	28-7
Setting MIRRORPATH and MIRROROFFSET with a Text Editor	28-7
Starting Mirroring for Unmirrored Dbspaces	28-8
Starting Mirroring for Unmirrored Dbspaces with ON-Monitor.	28-8
Starting Mirroring for Unmirrored Dbspaces with onspaces	28-8
Starting Mirroring for New Dbspaces	28-8
Starting Mirroring for New Dbspaces with ON-Monitor	28-8
Starting Mirroring for New Dbspaces with onspaces	28-9
Adding Mirrored Chunks	28-9
Adding Mirrored Chunks with ON-Monitor	28-9
Adding Mirrored Chunks with the onspaces Utility.	28-9
Changing the Mirror Status	28-10
Taking Down a Mirrored Chunk	28-10
Taking Down a Mirrored Chunk with ON-Monitor	28-10
Taking Down a Mirrored Chunk with the onspaces Utility	28-11
Recovering a Mirrored Chunk.	28-11
Recovering a Mirrored Chunk with ON-Monitor	28-11
Recovering a Mirrored Chunk with the onspaces Utility	28-11

Relinking a Chunk to a Device After a Disk Failure	28-12
Ending Mirroring	28-12
Ending Mirroring with ON-Monitor	28-13
Ending Mirroring with onspaces	28-13

T

his chapter describes the various mirroring tasks that are required to use the INFORMIX-Universal 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 Universal Server off-line and enable mirroring. See [“Enabling Mirroring” on page 28-4](#).
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 28-5](#).
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 28-6](#).

Enabling Mirroring

When you enable mirroring, you invoke the Universal 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 blob space. See [“Starting Mirroring” on page 28-6](#).

To enable mirroring for Universal 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 Universal 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 Universal Server is in on-line mode, but the change does not take effect until you reinitialize shared memory (take Universal Server off-line and then to quiescent or on-line mode).

If you are logged in as user **informix** or **root**, 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, select the Parameters menu, Initialize option. 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 Universal 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 Universal 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 Universal 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 15-4](#).

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 28-12](#).)

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, Universal Server performs the *recovery* process, copying data from the primary chunk to the mirrored chunk. When this process is complete, Universal Server begins mirroring data. If the primary chunk contains logical-log files, Universal 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 27-7](#).

You must always start mirroring for an entire dbspace or blobspace. Universal 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 Universal Server, you can also specify a mirror pathname and offset for the root chunk. Universal 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 27-7.](#))

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 wish to start mirroring the root dbspace, you must change the mirroring status of the root dbspace once Universal Server is initialized. See [“Starting Mirroring for Unmirrored Dbspaces” on page 28-8.](#)

Setting `MIRRORPATH` and `MIRROROFFSET` with ON-Monitor

If you are using ON-Monitor to initialize Universal 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 Universal Server, you must use a text editor to set the values of `MIRRORPATH` and `MIRROROFFSET` in the `ONCONFIG` file before you bring up Universal 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. Universal 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 Blobspaces, Dbspaces, Extspaces, or Sbspaces”](#) on page 39-55.

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 dspace. For example, the following command creates the dspace **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 Blobspaces, Dbspaces, Extspaces, or Sbspaces”](#) on page 39-55.

Adding Mirrored Chunks

If you add a chunk to a dspace 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 dspace. The following example adds a chunk, **chunk2**, to the **db_acct** dspace. Because the dspace 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 Blobspaces, Dbspaces, Extspaces, or Sbspaces”](#) on page 39-55.

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 Chunk Status” on page 33-55](#).

Taking Down a Mirrored Chunk

When a mirrored chunk is *down*, Universal 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 28-12](#).) Taking down a chunk is not the same as ending mirroring. You end mirroring for a complete dbspace, which causes Universal 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. Universal Server displays a screen that lists all the chunks in the dbspace. Move the cursor to the chunk that you wish to take down, and type 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 Blobspaces, Dbspaces, Extspaces, or Sbspaces”](#) on page 39-55.

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 the onspaces Utility

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 Blobspaces, Dbspaces, Extspaces, or Sbspaces”](#) on page 39-55.

Relinking a Chunk to a Device After a Disk Failure

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, Universal 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** 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 blob space with ON-Monitor, select the Mirror option of the Dbspaces menu. Select a dbspace or blob space 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 Blobspaces, Dbspaces, Extspaces, or Sbspaces”](#) on page 39-55.

What Is Data Replication?

What Is Data Replication?	29-3
What Is High-Availability Data Replication?.	29-4
What Are Primary and Secondary Database Servers?	29-4
How Is Data Replication Different from Mirroring?	29-6
How Is Data Replication Different from Two-Phase Commit? .	29-7
How Does Data Replication Work?	29-8
How Is the Data Initially Replicated?	29-8
Reproducing Updates to the Primary Database Server	29-9
How Are the Log Records Sent?.	29-9
What Are the Data-Replication Buffers?	29-10
When Are Log Records Sent?	29-10
Synchronous Updating.	29-11
Asynchronous Updating	29-11
What Threads Handle Data Replication?	29-13
Checkpoints Between Database Servers	29-14
How Is Data Synchronization Tracked?	29-14
Data-Replication Failures.	29-15
What Are Data-Replication Failures?	29-15
How Are Data-Replication Failures Detected?	29-16
What Happens When a Data-Replication Failure Is Detected?. . .	29-16
Considerations After Data-Replication Failure	29-17
Actions to Take If the Secondary Database Server Fails. . . .	29-17
Actions to Take If the Primary Database Server Fails	29-18
Redirection and Connectivity for Data-Replication Clients	29-22
Designing Clients for Redirection	29-22
Automatic Redirection with DBPATH	29-23
How Does the DBPATH Redirection Method Work?.	29-23
What Does the Administrator Need to Do?.	29-24
What Does the User Need to Do?	29-24

Administrator-Controlled Redirection with the sqlhosts File	29-24
How Does the sqlhosts File-Redirection Method Work?. . . .	29-24
What Does the Administrator Need to Do?	29-25
What Does the User Need to Do?	29-28
User-Controlled Redirection with INFORMIXSERVER	29-28
How Does the INFORMIXSERVER Redirection Method	
Work?	29-28
What Does the Administrator Need to Do?	29-29
What Does the User Need to Do?	29-29
Handling Redirection Within an Application.	29-29
A Connection Loop and Database Server Type Check	29-30
Comparison of Different Redirection Mechanisms	29-32
Designing Data-Replication Clients	29-33
Setting Lock Mode to Wait for Access to Primary Database Server .	29-33
Designing Clients to Use the Secondary Database Server	29-34
No Data Modification Statements	29-34
Locking and Isolation Level	29-36
Using Temporary Dbspaces for Sorting and Temporary Tables .	29-36

T

his chapter describes INFORMIX-Universal Server high-availability data replication. The following topics are covered:

- What data replication is, both in a broad sense and in the context of Universal Server
- How Universal Server data replication works
- How Universal Server data replication handles failures
- How the system administrator or user can redirect a client to connect to the other database server in the data-replication pair
- What the design considerations are for applications that connect to the secondary database server

A companion chapter, [Chapter 30, “Using Data Replication,”](#) contains instructions on how to accomplish the administrative tasks that are involved in using 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, Universal Server implements nearly transparent data replication of entire database servers. All the data managed by one Universal Server database server is replicated and dynamically updated on another Universal Server database server, often at a separate geographical location. Universal Server data replication is sometimes called *high availability* or *hot-site backup* because it 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 Universal Server database servers to use data replication, one Universal Server database server is called the *primary* database server, and the other is called the *secondary* database server. (In the context of data replication, a database server that does not use 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 Universal Server 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 29-1, the secondary Universal Server database server is dynamically updated, with changes made to the data managed by the primary database server.

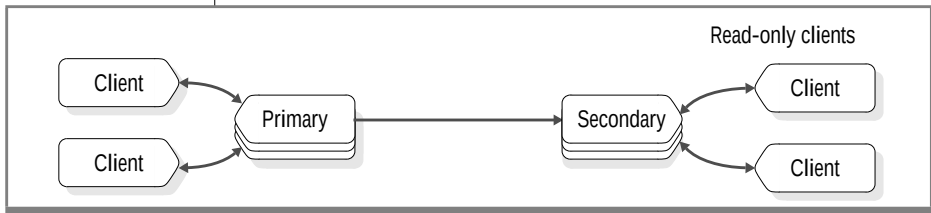


Figure 29-1
A Primary and Secondary Database Server in a Data-Replication Pair

If one of the database servers fails, as shown in Figure 29-2, you can redirect the clients that use that database server to the other database server in the pair.

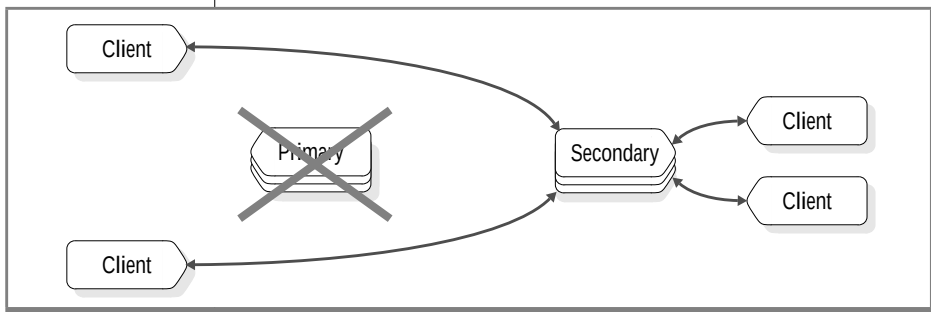


Figure 29-2
Database Servers in a Data-Replication Pair and Clients After a Failure

If a primary database server fails, you can change the secondary database server to a standard database server so that it can accept updates.

Universal Server 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

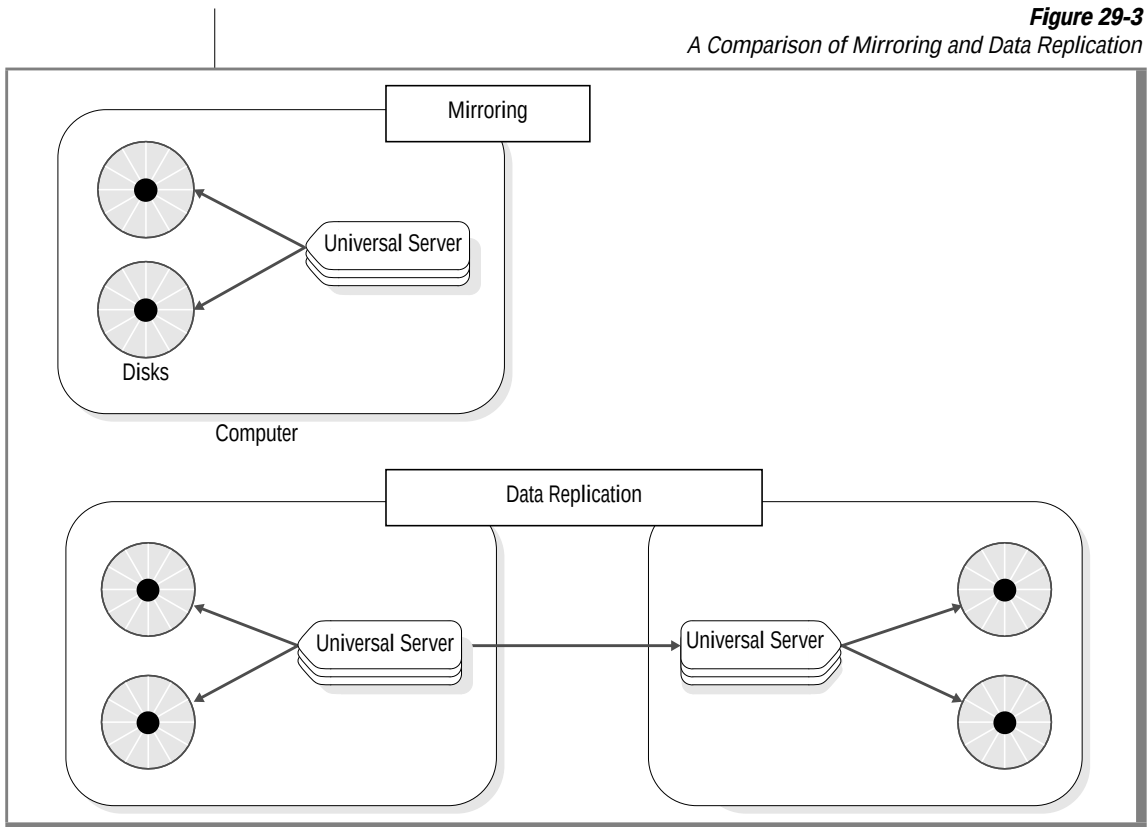
How Is Data Replication Different from Mirroring?

Universal Server data replication and mirroring are both transparent ways of making Universal Server more fault tolerant. However, as shown in [Figure 29-3 on page 29-7](#), they are quite different.

Mirroring, described in [“What Is Mirroring?” on page 27-4](#), is the mechanism by which a single Universal Server 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 Universal Server automatically updates data on both disks and automatically uses the other disk if one of the dbspaces fails.

Universal Server 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 Universal Server data replication involves two separate database servers, the data that these database servers manage is protected against all types of Universal Server failures, such as a computer crash or the catastrophic failure of an entire site, not just disk failures.

Figure 29-3
A Comparison of Mirroring and Data Replication



How Is Data Replication Different from Two-Phase Commit?

The two-phase commit protocol, described in detail in [Chapter 34, “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 Universal Server 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 data replication.

How Does Data Replication Work?

This section describes the mechanisms that Universal Server uses to perform data replication. For instructions on how to set up, start, and administer a data-replication system, refer to [Chapter 30, “Using Data Replication.”](#)

How Is the Data Initially Replicated?

Universal 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 Data Replication for the First Time” on page 30-10.](#)

You must perform the initial 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 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 Universal Server data replication is working, the primary database server is in on-line mode and accepts updates and queries just as a standard Universal Server 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

Universal Server 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 20-9](#).) The secondary database server receives the logical-log records generated on the primary database server and applies them to its dbspaces.



Important: Universal Server cannot replicate updates to databases that do not use transaction logging. Universal Server does not replicate data in blobspaces either.

How Are the Log Records Sent?

As shown in [Figure 29-4 on page 29-10](#), when the primary database server starts to flush the contents of the logical-log buffer in shared memory to the logical log on disk, Universal 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 Universal 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 Data-Replication Buffers?

The data-replication buffers are part of the virtual shared memory managed by the primary database server. The data-replication buffers hold logical-log records before the primary database server sends them to the secondary database server. The data-replication buffers are the same size as the logical-log buffers. Figure 29-4 shows this concept.

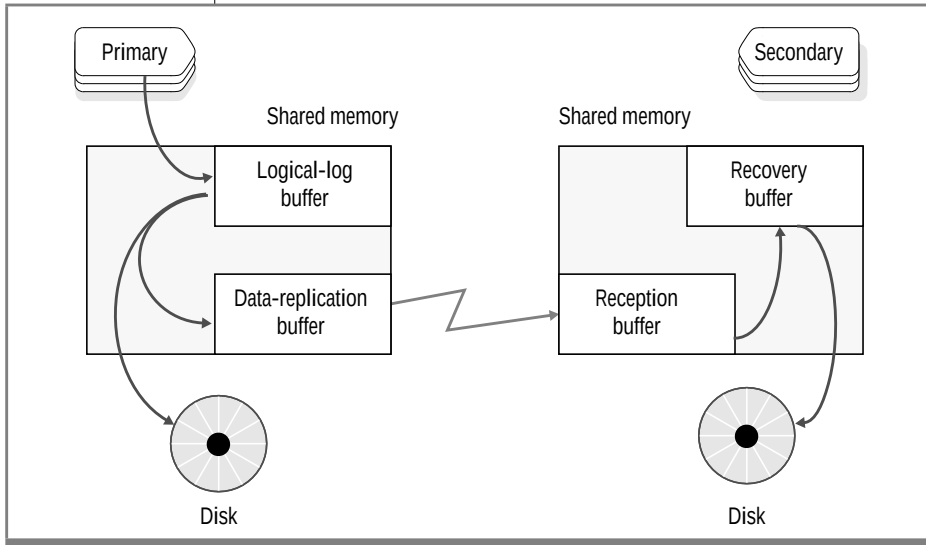


Figure 29-4
How Logical-Log Records Are Sent from the Primary Database Server to the Secondary Database Server

When Are Log Records Sent?

The primary database server sends the contents of the data-replication buffer to the secondary database server either *synchronously* or *asynchronously*. The value of the ONCONFIG configuration parameter DRINTERVAL, described on [page 37-19](#), determines whether Universal Server uses synchronous or asynchronous updating.

Synchronous Updating

If you set `DRINTERVAL` to `-1`, data replication occurs *synchronously*. As soon as the primary database server writes the logical-log buffer contents to the data-replication buffer, it sends those records from the 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 data-replication buffer. Independent of that action, the primary database server sends the contents of the data-replication buffer across the network when one of the following conditions occurs:

- The 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.

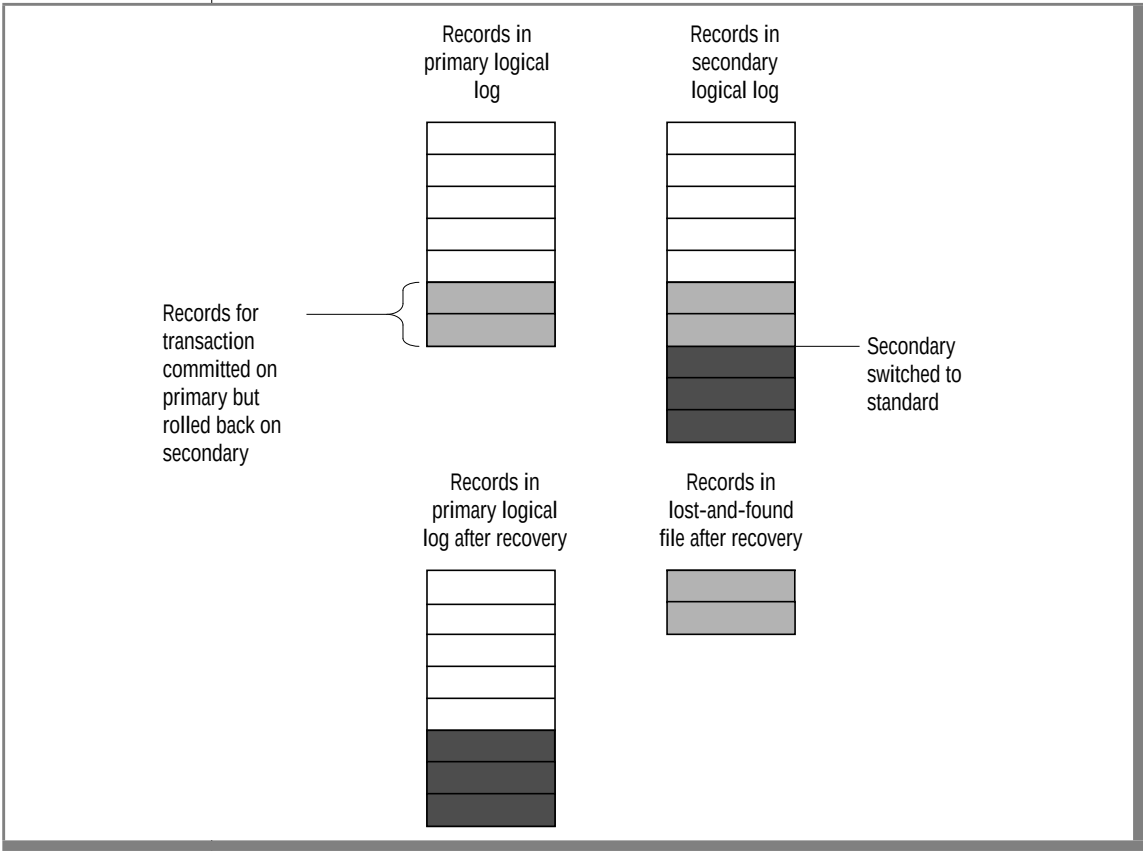
This method of updating might provide better performance than synchronous updating. However, as explained in the following section, transactions might be lost.

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 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, Universal 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 29-5 illustrates the process.

Figure 29-5
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. Universal 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 Data Replication?

Universal Server starts specialized threads to support data replication. As shown in Figure 29-6, a thread called **drprsend** on the primary database server sends the contents of the data-replication buffer across the network to a thread called **drsecrcv** on the secondary database server.

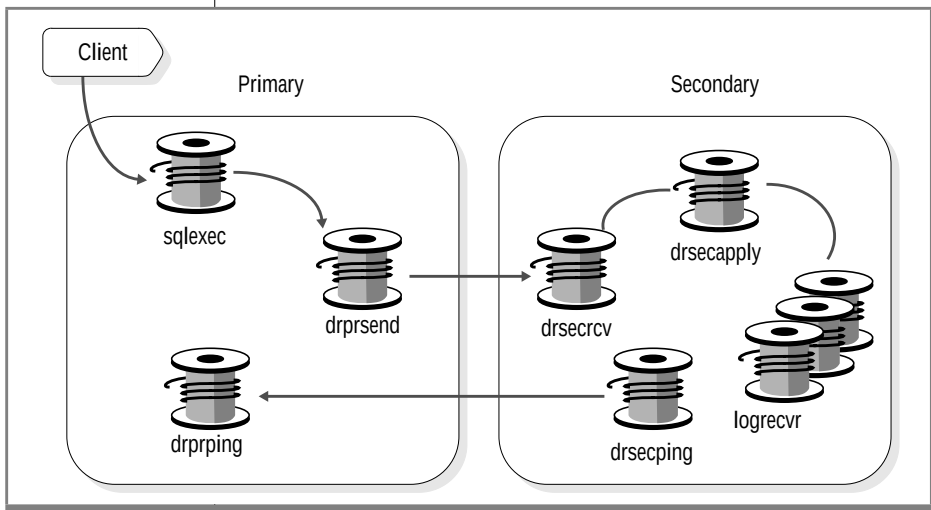


Figure 29-6
*Threads That
Manage Data
Replication*

A thread called **drsecapply** on the secondary Universal 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 Universal Server starts for 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 data-replication pair are synchronous, regardless of the value of DRINTERVAL. (See [“Checkpoints” on page 12-56](#).) 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 Data-Replication Failures?” on page 29-15](#).

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 42-14](#)):

- 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.

Data-Replication Failures

This section discusses the causes and consequences of a data-replication failure, as well as the administrator's options for managing failure and restarting data replication.

What Are Data-Replication Failures?

A data-replication failure is a loss of connection between the database servers in a 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 data-replication failure does not necessarily mean that one of the database servers has failed, only that the data-replication connection between the two database servers is lost.*

How Are Data-Replication Failures Detected?

The database server interprets either of the following conditions as a data-replication failure:

- A specified time-out value was exceeded.

In the course of normal 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 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 data-replication failure has occurred.

What Happens When a Data-Replication Failure Is Detected?

After a database server detects a data-replication failure, it writes a message to its message log (for example, DR: receive error) and turns data replication off. Thus, the data-replication connection between the two database servers is dropped. Both database servers experience the data-replication connection being dropped.

If the secondary database server remains on-line after a 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 29-18](#)). If DRAUTO is set to 0 (OFF), the secondary database server attempts to reestablish communication with the primary database server periodically.

Considerations After Data-Replication Failure

Consider the following two issues when a 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 29-22](#).

- How the database servers should react to the failure

Which administrative actions to take after a data-replication failure depend 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” on this page and [“Actions to Take If the Primary Database Server Fails” on page 29-18](#).

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 29-22](#). 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 30-30](#).

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 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 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 29-22.

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 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 29-20.

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 30-32.

When you succeed in bringing the original primary database server back on-line, the 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 29-7](#) on page 29-20.

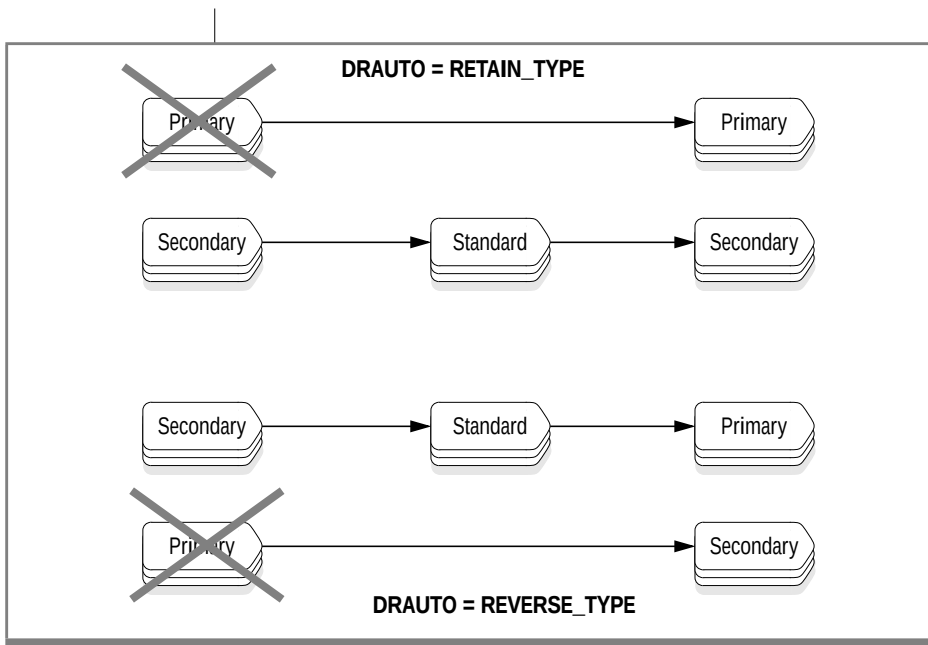


Figure 29-7
*Automatic
Switchover After a
Failure on the
Primary Database
Server*

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 Data Replication for the First Time” on page 30-10](#). 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 Database Server Type of the Secondary Database Server” on page 30-21](#).

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 30-31](#).

Restarting If the Secondary Database Server Is Not Switched to Standard Universal 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 30-31](#).

Redirection and Connectivity for Data-Replication Clients

Clients connect to the database servers in a 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.)

Universal Server does not have a transparent mechanism for directing client requests to different database servers in a data-replication pair, although you can automate this action from within the application as described in [“Handling Redirection Within an Application” on page 29-29](#).

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 **sqlhosts** file
- 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 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 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 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 sqlhosts File

This section explains the steps in redirecting clients with the **sqlhosts** file mechanism and the connectivity strategy that supports this method.

How Does the sqlhosts File-Redirection Method Work?

The **sqlhosts** file-redirection method relies on the fact that when an application connects to a database server, it uses information in the **sqlhosts** file to find that database server.

If one of the database servers in a data-replication pair is unusable, an administrator can change the definition of the unavailable database server in the **sqlhosts** file. 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 data-replication pair.

Because the **sqlhosts** file 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 **sqlhosts** file:

1. Change the **sqlhosts** file for the clients.
2. Change other connectivity files, if necessary.

These steps are described in the following sections. For information on the **sqlhosts** file, refer to [Chapter 4, “Client / Server Communications.”](#)

Changing the sqlhosts File

On the client computer, edit the **sqlhosts** file. 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 29-8 on page 29-27](#) shows how **sqlhosts** file entries might be modified to redirect clients.

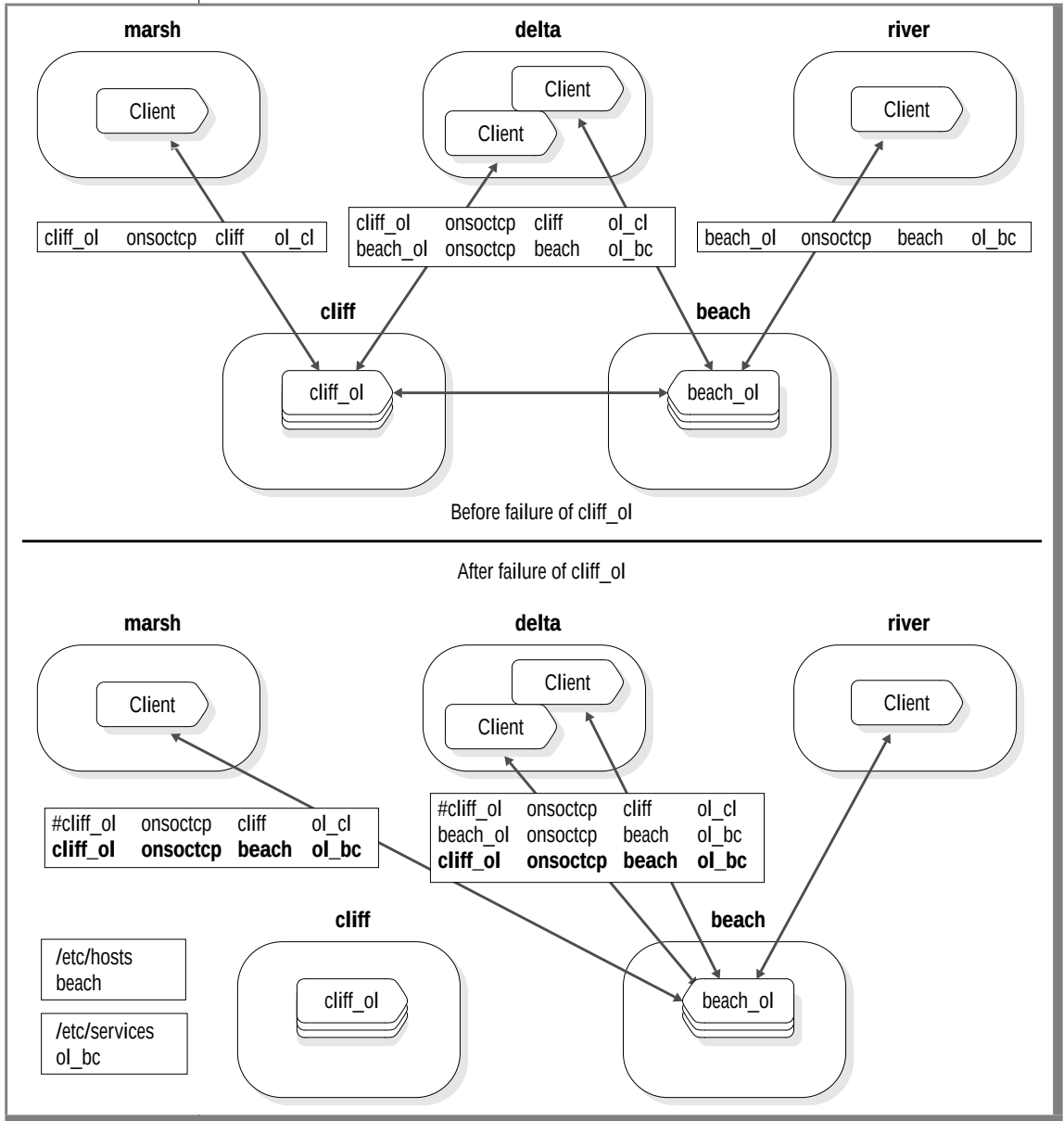
You do not need to change entries in the **sqlhosts** file on either of the computers that is running the database servers.

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 29-8
The sqlhosts File Entries Before and After a Failure of the cliff_ol Database Server



What Does the User Need to Do?

After the administrator changes the **sqlhosts** file 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 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 data-replication pair.
3. Restart their applications.

Handling Redirection Within an Application

If you use the **DBPATH** or **sqlhosts** file-redirection mechanism, you can include in your clients a routine that handles errors when clients encounter a 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 29-9 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 Universal 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 29-9
*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 **sqlhosts** file redirection method (explained in [“Administrator-Controlled Redirection with the sqlhosts File” on page 29-24](#)), 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 **sqlhosts** file 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 **sqlhosts** file or change the type of the secondary database server to standard).

Comparison of Different Redirection Mechanisms

Figure 29-10 summarizes the differences among the three redirection mechanisms.

Figure 29-10
Comparison of Redirection Methods for Different Connectivity Strategies

	DBPATH		sqlhosts file		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 sqlhosts file, 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 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 Universal Server performs a logical recovery, it normally defers index builds until the end of the recovery. However, if Universal 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 would fail *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 standard Universal Server database servers.
- 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 **sqlwarn6** of the **sqlwarn** field in the ESQL/C **sqlca** structure (and equivalent values for other SQL APIs). Universal Server sets **sqlwarn6** to W when runs it on 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 14-21](#) explains where Universal 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 15-9](#).
- 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.

Using Data Replication

Planning for Data Replication	30-3
Configuring Data Replication	30-4
Meeting Hardware and Operating-System Requirements	30-4
Meeting Database and Data Requirements	30-5
Meeting Database Server Configuration Requirements	30-6
Version	30-6
Dbospace and Chunk Configuration	30-6
Mirroring	30-7
Physical-Log Configuration	30-7
Dbospace and Logical-Log Tape Backup Devices	30-7
Logical-Log Configuration	30-8
Shared-Memory Configuration	30-8
Data-Replication Parameters	30-8
Configuring Data-Replication Connectivity	30-8
Starting Data Replication for the First Time	30-10
Performing Basic Universal Server Administration Tasks.	30-14
Changing Database Server Configuration Parameters	30-14
Dbospace and Logical-Log File Backups	30-15
Changing the Logging Status of Databases	30-16
Adding and Dropping Chunks, Dbspaces, and Blobspaces.	30-16
Using and Changing Mirroring of Chunks	30-16
Managing the Physical Log.	30-17
Managing the Logical Log	30-18
Managing Virtual Processors	30-18
Managing Shared Memory	30-18

Changing the Database Server Mode	30-19
Changing the Database Server Type	30-20
Changing the Database Server Type of the Primary Database Server	30-21
Changing the Database Server Type of the Secondary Database Server	30-21
Restoring Data If Media Failure Occurs	30-22
Restoring After Media Failure on the Primary Database Server . . .	30-22
Restoring After Media Failure on the Secondary Database Server . .	30-24
Restarting Data Replication After a Failure.	30-26
Restarting After Critical Data Is Damaged	30-26
Critical Media Failure on the Primary Database Server	30-26
Critical Media Failure on the Secondary Database Server . . .	30-28
Critical Media Failure on Both Database Servers	30-28
Restarting If Critical Data Is Not Damaged	30-28
Restarting After a Network Failure.	30-29
Restarting If the Secondary Database Server Fails.	30-30
Restarting If the Primary Database Server Fails	30-31

T

his chapter describes how to use INFORMIX-Universal Server data replication. If you plan to use data replication, read this entire chapter first. The following topics are covered:

- Planning for data replication
- Configuring a system for data replication
- Starting data replication
- Operating Universal Server database servers that use data replication
- Managing the mode of a database server in a data-replication pair
- Changing the type of a database server in a data-replication pair
- Restoring data after a media failure
- Managing data replication after a failure

A companion chapter, [Chapter 29, “What Is Data Replication?”](#) explains what data replication is, how it works, and how to design client applications for a data-replication environment.

Planning for Data Replication

Before you start setting up computers and database servers to use 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 Universal Server 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 21, “Managing Database-Logging Status.”](#)
- Develop client applications to make use of both database servers in the data-replication pair. For a discussion of design considerations, refer to [“Redirection and Connectivity for Data-Replication Clients” on page 29-22](#) and [“Designing Clients to Use the Secondary Database Server” on page 29-34.](#)
- Create a schedule for starting 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 data replication after a failure. Read [“Redirection and Connectivity for Data-Replication Clients” on page 29-22.](#)

Configuring Data Replication

To configure your system for 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 data-replication connectivity

Each of these topics is explained in this section.

Meeting Hardware and Operating-System Requirements

For a Universal Server data-replication database server pair to function, it must meet the following hardware requirements:

- The computers that run the primary and secondary Universal Server database servers must be identical (same vendor and architecture).
- The operating systems on the computers that run the primary and secondary Universal Server database servers must be identical.

- The hardware that runs the primary and secondary Universal Server database servers must support network capabilities.
- The amount of disk space allocated to nontemporary dbspaces for the primary and secondary Universal Server 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.

In addition, the Universal Server administrators of both the database servers should be able to communicate (for example, by telephone) when they perform administrative tasks.

Meeting Database and Data Requirements

For an Universal Server data-replication database server pair to function, you must meet the following database and data requirements:

- All databases that you wish to replicate must have transaction logging turned on.

This requirement is important because the secondary Universal Server database server uses logical-log records from the primary Universal Server database server to update the data 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 data replication, see either [“Turning On Transaction Logging with ON-Archive” on page 21-5](#) or [“Turning On Transaction Logging with ontape” on page 21-7](#).

- If your primary database server has simple large objects stored in blobspaces, modifications to the data within those blobspaces is not replicated as part of normal data-replication processing. simple large object data within dbspaces is replicated, however.

Meeting Database Server Configuration Requirements

For a data-replication database server pair to function, you must meet the following Universal Server database server configuration requirements.

To meet these requirements, you must fully configure each of the Universal Server database servers. For information on configuring a database server, refer to [“Configuring Universal Server” on page 3-12](#). You can then use the relevant aspects of that configuration to configure the other database server in the pair.

Version

The versions of Universal Server on the primary and secondary database servers must be Version 9.1 and must be identical.

Dbospace 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 Universal Server database servers.

The configuration must contain at least one temporary dbspace. See [“Using Temporary Dbspaces for Sorting and Temporary Tables” on page 29-36](#).

You can use symbolic links for the chunk pathnames, as explained in [“Creating Links to Each Raw Device” on page 15-7](#).

The following ONCONFIG parameters must have the same value on each database server:

- ROOTNAME (see [page 37-60](#))
- ROOTPATH (see [page 37-61](#))
- ROOTOFFSET (see [page 37-61](#))
- ROOTSIZE (see [page 37-62](#))

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:

- MIRRORPATH (see [page 37-45](#))
- MIRROROFFSET (see [page 37-45](#))

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 37-57](#))
- PHYSFILE (see [page 37-57](#))

Dbospace 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 dbospace and logical-log tape backup devices should be identical. The following ONCONFIG parameters must have the same value on each database server:

- TAPEBLK (see [page 37-70](#))
- TAPESIZE (see [page 37-73](#))
- LTAPEBLK (see [page 37-38](#))
- LTAPESIZE (see [page 37-40](#))

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 37-34](#))
- LOGSIZE (see [page 37-35](#))

Shared-Memory Configuration

Set all the shared-memory configuration parameters to the same values on the two database servers.

Data-Replication Parameters

The following parameters are specific to data replication and must be set to the same value on both database servers in the data-replication pair:

- DRINTERVAL (see [page 37-19](#))
- DRLOSTFOUND (see [page 37-20](#))
- DRTIMEOUT (see [page 37-20](#))
- DRAUTO (see [page 37-18](#))

Configuring Data-Replication Connectivity

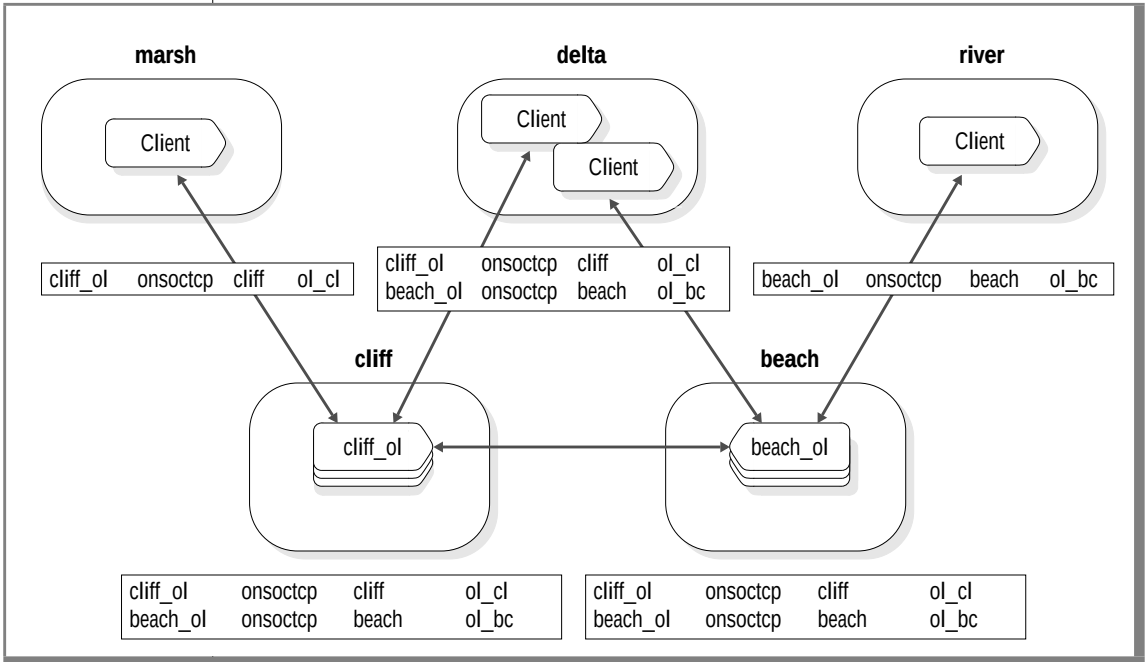
For a Universal Server data-replication database server pair to function, the database servers in the data-replication pair must be able to establish a connection with one another. To satisfy this requirement, the **sqlhosts** file on each of the computers that is running Universal Server in a data-replication pair must have at least the following entries:

- An entry that identifies the Universal Server database server that is running on that computer
- An entry that identifies the other Universal Server database server in the data-replication pair

Figure 30-1 shows a sample data-replication configuration and example **sqlhosts** file entries necessary for data replication.

Figure 30-1

Example sqlhosts File Entries for Database Servers in a Data-Replication Pair



In addition to entries in the **sqlhosts** files, the computers that are running Universal Server in a data-replication pair must have entries for the other computer and service in their **/etc/hosts** and **/etc/services** files.

Starting Data Replication for the First Time

After you complete data-replication configuration, you are ready to start data replication. This section describes the necessary steps for starting data replication.

Suppose you wish to start data replication on two database servers, **ServerA** and **ServerB**. The procedure for starting data replication, using **ServerA** as the primary database server and **ServerB** as the secondary database server, is described in the following steps. [Figure 30-2 on page 30-12](#) 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 30-2](#) also shows messages sent to the message log.

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 data-replication connection with the other database server in the data-replication pair and to start 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 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 data-replication connection with the primary database server (**ServerA**) and start operation. The connection should be successfully established.

Before 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 30-2
Steps in Starting 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/TAP E=(primary:/dev/remotedrive) ontape command % ontape -p Answer no when prompted to back up the logs. Messages to message log INFORMIX-Universal 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 <pre>% onmode -d secondary prim_name</pre> Messages to message log <pre>DR: new type = secondary, primary server name = prim_name</pre> 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).
5	Messages to message log <pre>DR: Primary server connected DR: Primary server operational</pre>	Messages to message log <pre>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_RECVR_THREADS'). Logical Recovery Started Start Logical Recovery - Start Log n, End Log? Starting Log Position - n 0xn timer DR: Secondary server operational</pre> ON-Archive command <pre>ONDATARTR> RETRIEVE/LOGFILE/TAPE=(primary:/dev/r emotedevice)</pre>

(2 of 3)

Step	On the Primary	On the Secondary
	Messages to message log DR: Primary server connected DR: Primary server operational	ontape command % ontape -l 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 Universal Server Administration Tasks

This section contains instructions on how to perform basic Universal Server administration tasks once your system is running data replication.

Changing Database Server Configuration Parameters

Some of the Universal Server configuration parameters must be set to the same value on both database servers in the data-replication pair (as listed under [“Meeting Database Server Configuration Requirements” on page 30-6](#)). Other Universal 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 30-1 on page 30-9](#) and [Figure 30-2 on page 30-12](#) 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 data-replication pair, you can change the value on the primary or secondary database server individually.

Dbospace and Logical-Log File Backups

When you use data replication, you must back up logical-log files and create dbospace backups of your data, just as you would with a standard Universal Server database server. You need to perform dbospace and logical-log file backups only on the primary database server. Be prepared, however, to perform dbospace 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 dbospace backups and logical-log backups) must be identical on the primary and secondary Universal Server database servers.

Changing the Logging Status of Databases

You cannot add transaction logging to databases on the primary database server while you are using 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 data replication off, add logging, and then perform a dbspace backup and restore as described in [“Starting Data Replication for the First Time” on page 30-10](#).

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 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 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 15-4](#).

If you want to mirror a dbspace on one of the database servers in the 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 15-4](#).

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 28-11](#).

When you drop a chunk from the primary database server, Universal 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, Universal 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 **onspace -r**. For more information on turning mirroring off, see [“Ending Mirroring” on page 28-12](#).

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 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 30-14](#).

For information on changing the size and location of the physical log, refer to [Chapter 25, “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 23, “Managing Logical-Log Files.”](#) Universal 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 30-14.](#)

Changing the Database Server Mode

The effects of changing the mode of a database server in a data-replication pair differ depending on whether you are changing the mode of the primary or the secondary database server.

Figure 30-3 summarizes the effects of changing the mode of the primary database server.

Figure 30-3
Mode Changes on the Primary Database Server

On the Primary	On the Secondary	To Restart Data Replication
Any mode → off-line (onmode -k)	Secondary receives errors. 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 30-31 ■ “The Secondary Database Server Is Changed to a Standard Database Server Manually” on page 30-31 ■ “The Secondary Database Server Is Changed to a Standard Database Server Automatically” on page 30-32
On-line → quiescent (onmode -s / onmode -u)	Secondary does not receive errors. Data replication remains on. Mode remains read-only.	Use onmode -m on the primary.

Figure 30-4 summarizes the effects of changing the mode of the secondary database server.

Figure 30-4
Mode Changes on the Secondary Database Server

On the Secondary	On the Primary	To Restart Data Replication
Read-only → off-line (onmode -k)	Primary receives errors. 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 30-30.

Changing the Database Server Type

You might want to stop the 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 30-19). 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 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 39-35.

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 data-replication pair does not change, but data replication is turned off.

Changing the Database Server 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 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, data replication is turned off on the secondary database server.

To change the database server back to type primary and restart data replication, execute the following command:

```
% onmode -d primary secondary
```

Changing the Database Server 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 data replication, you must follow the entire procedure in [“Starting Data Replication for the First Time” on page 30-10](#).

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 30-5 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 30-26](#).

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, refer to the [INFORMIX-Universal Server Backup and Restore Guide](#). For more information on performing a warm restore using ON-Archive, refer to the [INFORMIX-Universal Server Archive and Backup Guide](#). You must back up all logical-log files before you perform the warm restore.

Figure 30-5
Different Scenarios for Media Failure on the Primary Database Server

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 30-26.
Primary	Yes	Yes	Primary database server remains on-line. Follow the procedures in “Recovering a Mirrored Chunk” on page 28-11.
Primary	No	No	Primary database server remains on-line. Follow the procedure in your Universal 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 28-11.

Restoring After Media Failure on the Secondary Database Server

[Figure 30-6 on page 30-25](#) 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 30-26.](#)) 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 30-6
Different Scenarios for Media Failure on the Secondary Database Server

Data-Replication Server	Critical Media?	Chunks Mirrored?	Effect of Failure
Secondary	Yes	No	Secondary database server fails. Primary database server receives errors. Data replication is turned off. Follow the procedure in “Restarting After Critical Data Is Damaged” on page 30-26
Secondary	Yes	Yes	Secondary database server remains on-line in read-only mode. Follow the procedures in “Recovering a Mirrored Chunk” on page 28-11.
Secondary	No	No	Secondary database server remains on-line in read-only mode. Primary database server receives errors. Data replication is turned off. Follow the procedure in “Restarting After Critical Data Is Damaged” on page 30-26.
Secondary	No	Yes	Secondary database server remains on-line in read-only mode. Follow the procedures in “Recovering a Mirrored Chunk” on page 28-11.

Restarting Data Replication After a Failure

[“What Are Data-Replication Failures?” on page 29-15](#) discusses the various types of data-replication failure. The procedure that you must follow to restart 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 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 data replication after the primary database server suffers a critical media failure, perform the following steps. Figure 30-7 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 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, 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 30-7

Steps for Restarting Data Replication After a Critical Media Failure on the Primary Database Server

Step	On the Primary	On the Secondary
1		onmode command % onmode -s % onmode -d secondary <i>prim_name</i>
2	ontape command % ontape -p ON-Archive command ONDATARTR> RETRIEVE/DBSPACESET=*/REQUEST=ri d/TAPE=(primary:/dev/remotedrive)	
3	onmode command % onmode -d primary <i>sec_name</i>	
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 Data Replication for the First Time” on page 30-10](#).

Critical Media Failure on Both Database Servers

In the unfortunate event that both of the computers that are running database servers in a 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 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 data replication for the first time. (See [“Starting Data Replication for the First Time” on page 30-10](#)). 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 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. Data replication is turned off on both database servers (state = off). When the connection is reestablished, you can restart data replication by issuing `onmode -d secondary primary_name` on the secondary database server. Restarting 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 29-20.](#)) In this case, you must follow the procedure discussed in [“Starting Data Replication for the First Time” on page 30-10.](#)

Restarting If the Secondary Database Server Fails

If you need to restart data replication after a failure of the secondary database server, complete the steps in Figure 30-8. 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 30-8
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 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 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 on-line using **oninit**.

The Secondary Database Server Is Changed to a Standard Database Server Manually

If you need to restart 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 30-9.

Figure 30-9
Steps to Restart If You Manually Changed the Secondary Database Server to a Standard Database Server

Step	On the Primary	On the Secondary
1		<pre>% onmode -s</pre> 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 29-22.
2		<pre>% onmode -d secondary prim_name</pre>

(1 of 2)

Restarting If Critical Data Is Not Damaged

Step	On the Primary	On the Secondary
3	<code>% oninit</code> 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 <pre>Onarchive> CATALOG/VSET=remote_logs/VOLUME= volnum/SID=sysid Onarchive>RETRIEVE/LOGFILE/VSET= remote_logs</pre> ontape command <pre>% ontape -l</pre>	

(2 of 2)

The Secondary Database Server Is Changed to a Standard Database Server Automatically

If you need to restart 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 29-18](#)), complete the steps in [Figure 30-10 on page 30-33](#).

Figure 30-10
Steps to Restart If You Automatically Changed the Secondary Database Server to a Standard Database Server

Step	On the Primary	On the Secondary
1	<code>% oninit</code> 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 29-22. 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.
2	If you are prompted to recover logical-log records from tape, perform this step. ON-Archive command <pre>Onarchive> CATALOG/VSET=remote_logs/VOLUME=volnum/S ID=sysid Onarchive>RETRIEVE/LOGFILE/VSET=remote_logs</pre> ontape command <pre>% ontape -l</pre>	

What Is Consistency Checking?

Performing Periodic Consistency Checking	31-3
Verify Consistency.	31-4
oncheck -cr	31-4
oncheck -cc	31-5
oncheck -ce	31-5
oncheck -cI	31-5
oncheck -cs	31-6
oncheck -cD	31-6
Monitor for Data Inconsistency	31-6
Retain Consistent Level-0 Dbspace	31-8
Dealing with Corruption	31-8
Symptoms of Corruption	31-9
Run oncheck First	31-9
I/O Errors on a Chunk	31-10
Collecting Diagnostic Information	31-10

INFORMIX-Universal Server is designed to detect Universal 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, Universal Server reports an *assertion failure* and writes text that describes the check that failed into the Universal Server message log. Universal 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 Universal 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 -cs**
- **oncheck -cI** *dbname*
- **oncheck -cD** *dbname*

The following sections describe these commands.

You can run each of these commands while Universal 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 39-7](#).

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.

oncheck -cr

Execute **oncheck -cr** to validate the Universal Server reserved pages that reside at the beginning of the initial chunk of the root dbspace. These pages contain the primary Universal Server overhead information. If this command detects errors, perform a data restore from dbspace backup. For information on how to restore your data from a dbspace backup, refer to the [INFORMIX-Universal Server Backup and Restore Guide](#), if you use ON-Bar for your backup and restore tool. If you use ON-Archive or **ontape** for your backup and restore tool, refer to the [INFORMIX-Universal Server Archive and Backup Guide](#).

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 Universal 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 Universal Server database. Extents must not overlap. If this command detects errors, perform a data restore from dbspace backup. If this command detects errors, perform a data restore from dbspace backup. If you use ON-Bar for your backup and restore tool, refer to the [INFORMIX-Universal Server 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, refer to the [INFORMIX-Universal Server Archive and Backup Guide](#).

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 -cs

Execute **oncheck -cs** for each database to validate metadata for all smart large objects in a Universal Server database. If necessary, restore the data from a dbspace backup. For information on how to restore your data from a dbspace backup, refer to the [INFORMIX-Universal Server Backup and Restore Guide](#), if you use ON-Bar for your backup and restore tool. If you use ON-Archive or **ontape** as your backup and restore tool, refer to the [INFORMIX-Universal Server Archive and Backup Guide](#).

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, refer to the [Informix Migration Guide](#). If this procedure does not succeed, perform a data restore from dbspace backup. If you use ON-Bar for your backup and restore tool, refer to the [INFORMIX-Universal Server Backup and Restore Guide](#) for information on how to restore your data from a dbspace backup. If you use ON-Archive or **ontape** as your backup and restore tool, refer to the [INFORMIX-Universal Server Archive and Backup Guide](#).

Monitor for Data Inconsistency

If the consistency-checking code detects an inconsistency during Universal Server operation, an assertion failure is reported to the Universal Server message log. (See [“What Is the Message Log?”](#) on page 33-6.)

Figure 31-1 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 Universal Server entity
Action: What action the Universal Server administrator should take
See Also: file(s) containing additional diagnostics
```

Figure 31-1
*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**

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 DUMPPDIR.

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 Universal Server shared memory at the time of the assertion failure, but only if the ONCONFIG parameter DUMPSHMEM is set to 1.

The file **gcore.xxx** contains a core dump of the Universal 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 Universal 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 Universal 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, Universal Server stops immediately when an assertion fails. However, when failures appear to be specific to a table or smaller entity, Universal Server continues to run.

When an assertion fails because of inconsistencies on a data page that Universal 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 40, “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 31-4](#) 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 Universal Server was verified to be consistent.

Dealing with Corruption

This section describes some of the symptoms of Universal Server system corruption and actions that Universal Server or you, as administrator, can take to resolve the problems. Corruption in a Universal Server database can occur as a consequence of hardware or operating-system problems, or from some unknown Universal Server problems. Corruption can affect either data or Universal Server overhead information.

Symptoms of Corruption

Universal 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 Universal 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 31-6](#).
- 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 Universal 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 (Universal 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 31-4](#).

I/O Errors on a Chunk

If an I/O error occurs during Universal 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 Universal 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, Universal Server continues to operate using the mirrored chunk. Use operating-system utilities to determine what is wrong with the down chunk and then to correct the problem. You must then direct Universal Server to restore mirrored chunk data. For instructions on how to recover a mirrored chunk, see [“Recovering a Mirrored Chunk” on page 28-11](#).

If the down chunk is not mirrored and contains logical-log files, the physical log, or the root dbspace, Universal Server immediately initiates an abort. Otherwise, Universal 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 Universal Server to off-line mode when a chunk is marked as down (D), you can reinitialize Universal Server provided that the chunk marked as down does not contain critical media (logical-log files, the physical log, or the root dbspace).

Collecting Diagnostic Information

Universal Server facilitates the collection of diagnostic information through the setting of several ONCONFIG parameters. 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 **memory/gcore/core dumps** 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.

The following ONCONFIG parameters direct Universal Server to preserve diagnostic information whenever an assertion failure is detected or whenever Universal Server enters into an abort sequence:

■ DUMPCORE	page 37-27
■ DUMPGCORE	page 37-28
■ DUMPSHMEM	page 37-29
■ DUMPDIR	page 37-28
■ DUMPCNT	page 37-27

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.*

Universal 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.

Situations to Avoid

Situations to Avoid in Administering the Database Server 32-3

O

ccasionally, INFORMIX-Universal Server administrators conceive of a shortcut that seems like a good idea. Because of the complexity of Universal Server, an idea that appears to be an efficient time saver can create problems elsewhere during operation. This chapter attempts to protect you from bad ideas that sound good.

The *[INFORMIX-Universal Server Backup and Restore Guide](#)* provides advice on situations to avoid when using ON-Bar to back up or restore Universal Server data. The *[INFORMIX-Universal Server Archive and Backup Guide](#)* provides advice on situations to avoid when you use ON-Archive or **ontape** 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 Universal Server performance:

- Never make changes to the tables in the **sysmaster** database.
- Never kill a Universal Server process (virtual processor) because killing a process causes Universal Server to terminate.
- Do not run more CPU virtual processors than you have CPUs in your hardware configuration.
- Never bring on-line two different types of network services with the same service name. For information on the service name, see “[The \\$INFORMIXDIR/etc/sqlhosts File](#)” on page 4-24.
- Avoid transactions that span a significant percentage of available logical-log space. See the descriptions of the LTXHWM and LTXEHWM parameters in [Chapter 37, “Configuration Parameters.”](#)

- 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 (using the **&** operator).
- Do not move a chunk from one dbspace or blob space to another without performing 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 INFORMIX- Universal Server

Section VII



Monitoring the Database Server

Sources of Information for Monitoring the Database Server	33-6
What Is the Message Log?	33-6
Why Read the Message Log?.	33-6
Changing Where Message-Log Messages Are Sent	33-7
Monitoring the Message Log.	33-7
Event Alarm	33-8
What Is the Console?.	33-12
Changing Where Console Messages Appear	33-12
Monitoring with ON-Monitor.	33-12
Monitoring with SMI Tables	33-13
Monitoring with onstat and oncheck Utilities	33-13
Comparing oncheck to onstat	33-13
Monitoring with onperf	33-13
Monitoring with the onstat Banner Line	33-14
Monitoring Configuration Information	33-15
Using Command-Line Utilities	33-15
Using ON-Monitor	33-16
Monitoring Checkpoint Information	33-17
Using Command-Line Utilities	33-17
Using ON-Monitor	33-18
Using SMI Tables.	33-18
Monitoring Shared Memory.	33-19
Monitoring Shared-Memory Segments.	33-19
Using Command-Line Utilities	33-19
Monitoring Shared-Memory Profile.	33-20
Using Command-Line Utilities	33-20
Using ON-Monitor	33-20
Using SMI Tables.	33-21

Monitoring Buffers	33-21
Using Command-Line Utilities	33-21
Using ON-Monitor	33-24
Using SMI Tables	33-24
Monitoring Buffer-Pool Activity	33-24
Using Command-Line Utilities	33-25
Using SMI Tables	33-27
Monitoring Latches	33-27
Using Command-Line Utilities	33-27
Using ON-Monitor	33-28
Using SMI Tables	33-28
Monitoring Locks	33-29
Using Command-Line Utilities	33-29
Using ON-Monitor	33-30
Using SMI Tables	33-31
Monitoring Active Tblspaces	33-31
Using Command-Line Utilities	33-31
Monitoring Virtual Processors	33-32
Using Command-Line Utilities	33-32
Using SMI Tables	33-34
Monitoring Sessions and Threads	33-35
Using Command-Line Utilities	33-35
Using ON-Monitor	33-38
Using SMI Tables	33-39
Monitoring Parallel Database Query	33-40
Using Command-Line Utilities	33-40
Monitoring Transactions	33-45
Using Command-Line Utilities	33-45
Monitoring Databases	33-47
Using ON-Monitor	33-47
Using SMI Tables	33-47
Monitoring Logging Activity	33-48
Monitoring Logical-Log Files	33-48
Monitoring the Logical Log for Fullness	33-48
Using Command-Line Utilities	33-49
Using ON-Monitor	33-50
Using SMI Tables	33-51

Monitoring the Physical-Log File	.33-51
Using Command-Line Utilities.	.33-51
Using ON-Monitor	.33-52
Monitoring the Physical-Log and Logical-Log Buffers	.33-53
Using Command-Line Utilities.	.33-53
Using ON-Monitor	.33-54
Using SMI Tables	.33-54
Monitoring Chunk Status	.33-55
Using Command-Line Utilities.	.33-55
Using ON-Monitor	.33-57
Using SMI Tables	.33-58
Monitoring the Database Server for Disabling I/O Errors	.33-58
Using the Message Log to Monitor Disabling I/O Errors	.33-58
Using Event Alarms to Monitor Disabling I/O Errors	.33-59
Monitoring Disk Usage	.33-60
Monitoring Chunks	.33-60
Using Command-Line Utilities.	.33-60
Using ON-Monitor	.33-63
Using SMI Tables	.33-64
Monitoring Tblspaces and Extents	.33-64
Using Command-Line Utilities.	.33-64
Using SMI Tables	.33-66
Using System Catalog Tables	.33-67
Monitoring Sbspaces	.33-67
Using Command-Line Utilities.	.33-67
Monitoring Simple Large Objects in a Blobospace	.33-69
Using Command-Line Utilities.	.33-69
Using ON-Monitor	.33-73
Monitoring Simple Large Objects in a Dbospace	.33-74
Using Command-Line Utilities.	.33-74
Monitoring Data-Replication Status.	.33-76
Using Command-Line Utilities.	.33-76
Using ON-Monitor	.33-77
Using SMI Tables	.33-78

T

he first part of this chapter describes sources of information available for monitoring INFORMIX-Universal Server.

The remainder of the chapter describes how to monitor the following different aspects of Universal Server:

- Universal 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)
- Databases
- Logging activity
- Disk usage
 - Chunks
 - Tblspaces and extents
 - Simple large objects in a blob space
 - Simple large objects in a dbspace
- Data-replication information

Sources of Information for Monitoring the Database Server

The following Universal Server tools and log files provide information about its activity and data:

- The message log
- The system console
- ON-Monitor
- 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 Universal Server *message log* is an operating-system file. The messages contained in the Universal Server message log do not usually require immediate action. To report situations that require your immediate attention, Universal Server uses the event-alarm feature. See [“Event Alarm” on page 33-8](#).

To specify the message-log pathname, set the value of MSGPATH in the ONCONFIG file. See [“MSGPATH” on page 37-46](#).

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 Universal Server processing. The messages are listed in [Chapter 40, “Message-Log Messages.”](#)

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

If you wish, you can read the Universal Server message log for a minute-by-minute account of Universal Server processing in order to catch events before a problem develops. However, Informix does not expect you to do this kind of monitoring.

Changing Where Message-Log Messages Are Sent

You can change the value of MSGPATH while Universal Server is in on-line mode, but the changes do not take effect until you reinitialize shared memory.

You must be logged in as user **informix** or user **root** to change the value of MSGPATH. You can make this change from within ON-Monitor or from the command line.

Using ON-Monitor to Change the Message Log

Select the Parameters menu, diaGnostics option to change the console message destination. In the field labelled **Message Log**, enter the pathname. Press ESC to record changes.

When the prompt appears to confirm that you want to save the changes to your configuration, respond Y (yes).

Using a Text Editor to Change the Message Log

To change the value of MSGPATH from the command line, use a text editor to edit the ONCONFIG file.

Monitoring the Message Log

Monitor the message log periodically to verify that Universal Server operations are proceeding normally and that events are being logged as expected. Use a UNIX editor to read the complete message log.

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

Using onstat -m to Read the Message Log

Execute the **onstat -m** command to obtain the name of the Universal Server message log and the 20 most-recent entries.

Event Alarm

Universal Server provides a mechanism for automatically triggering administrative actions based on an event that occurs in the Universal Server environment. This mechanism is the event-alarm feature.

To use the event-alarm feature, you must 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 33-1 for values)	integer
Event class ID (see Figure 33-2 for values)	integer
Event class msg (see Figure 33-2 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 33-2 on page 33-10](#) 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
OnLine 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 33-1. Message log events that have severity 1 do not cause the database server to invoke the alarm program.

Figure 33-1
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 33-2.

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 33-2.

Figure 33-2
Class-ID and Class-Message Values

Class ID	Class Message
1	Table failure: '%s' (dbname:"owner".tablename)
2	Index failure: '%s' (dbname:"owner".tablename-idxname)
3	Simple large object failure: '%s' (dbname:"owner".tablename)
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	Universal 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	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	Universal 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

Universal 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?

Universal 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 value of `CONSOLE` in the `ONCONFIG` file. See [“CONSOLE” on page 37-12](#).

Changing Where Console Messages Appear

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.

You must be logged in as user **informix** or user **root** to change the value of `CONSOLE`. You can make this change from within ON-Monitor or from the command line.

Using ON-Monitor to Change Console

Select the Parameters menu, `diagnostics` option to change the console-message destination. In the field labelled `Console Msgs.`, enter the pathname. Press `ESC` to record changes.

When the prompt appears to confirm that you want to save the changes to your configuration, respond `Y` (yes).

Using a Text Editor to Change Console

To change the value of `CONSOLE` from the command line, use a text editor to edit the `ONCONFIG` file.

Monitoring with ON-Monitor

ON-Monitor provides a simple way to monitor many aspects of Universal Server. Most of the monitoring functions are available under the Status menu. See [Chapter 36, “ON-Monitor.”](#)

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 38-5](#).

Monitoring with onstat and oncheck Utilities

The **onstat** and **oncheck** utilities provide a way to monitor database-server information from the command line.

Comparing oncheck to onstat

You can compare the display options of the **oncheck** utility to the **onstat** utility. 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

Universal 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**:

- The **onperf** utility displays the values of the metrics graphically in real time.
- The **onperf** utility allows you to choose which metrics to monitor.
- The **onperf** utility saves recent-history metrics data to a buffer in memory. This data is available if you want to analyze a recent trend.
- The **onperf** utility has the ability to save performance data to a file.

For more information on the **onperf** tool, refer to the [INFORMIX-Universal Server Performance Guide](#).

Monitoring with the onstat Banner Line

Whenever the database server is blocked, **onstat** displays the following additional 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	Universal Server failure
DBS_DROP	dropping a dbspace
DDR	discrete 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 33-48](#).

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

onstat -c

Execute **onstat -c** to display a copy of the configuration file. The values of the configuration parameters are stored in the file indicated by the **ONCONFIG** environment variable or, if you have not set the **ONCONFIG** environment variable, in **\$INFORMIXDIR/etc/onconfig**.

Changes to the configuration 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 **ONCONFIG** is not set, the database server displays the contents of the file **\$INFORMIXDIR/etc/onconfig**.

oncheck -pr

Execute **oncheck -pr** to obtain the configuration information that Universal 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 33-3. 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-Universal Server reserved pages -  
PAGE_CONFIG  
  ROOTNAME                rootdbs  
  ROOTPATH                /home/online/root_chunk  
  ROOTOFFSET              0  
  ROOTSIZE                8000  
  MIRROR                  0  
  MIRRORPATH  
  MIRROROFFSET            0  
  PHYSDBS                 rootdbs  
  PHYSFILE                1000  
  LOGFILES                5  
  LOGSIZE                 500  
  MSGPATH                 /home/online/online.log  
  CONSOLE                  /dev/ttyp5  
.  
. .  
. .  
.
```

Figure 33-3
oncheck -pr
PAGE_CONFIG
Output

Using ON-Monitor

Select the Status menu, Configuration option. 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 by default.

If you modify the ONCONFIG parameters but have not yet reinitialized shared memory, the effective parameters might be different than the parameters that appear in the file specified by ONCONFIG.

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, refer to the [INFORMIX-Universal Server 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 UNIX 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.

Using ON-Monitor

You can use the following ON-Monitor options to monitor checkpoint information.

Profile Option

Select the Status menu, Profile option. 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 33-4 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 33-4
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

Query the **sysprofile** table to obtain the same checkpoint statistics that are available from the **onstat -p** utility and the ON-Monitor Profile option. This 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 <i>critical section</i> 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 use the **onstat -o** utility to capture a static snapshot of Universal 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. Universal 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 37-67](#).

Using Command-Line Utilities

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 33-5.

```
RSAM Version 9.10.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 33-5
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 Universal Server activity. These statistics are shown in Figure 33-6.

```
RSAM Version 9.10.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

idxda-RA idx-RA  da-RA   RA-pgsused lchwaits
5         0       204     148        12
```

Figure 33-6
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, refer to [“-p Option” on page 39-96](#).

Using ON-Monitor

Select the Status menu, Profile option. 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 using this statistic to tune Universal Server, see the [INFORMIX-Universal Server Performance Guide](#).) The number of waits for buffers gives a measure of system concurrency.

Information on specific buffers includes a listing of all the buffers in shared memory that are held by a thread. This information allows you to track the status of a particular buffer. For example, you can determine if another thread is waiting for the buffer.

Using Command-Line Utilities

You can use the following command-line utilities to monitor buffers.

onstat -p

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 33-7 shows these fields.

```
RSAM Version 9.10.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 33-7
*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 33-8.

```
RSAM Version 9.10.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 33-8
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 33-9. For more information on the fields displayed by this option, refer to “[-b Option](#)” on page 39-83.

```
RSAM Version 9.10.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 33-9
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.

Using ON-Monitor

To access the fields mentioned on [page 33-21](#) for **onstat -p (bufreads, %cached, bufwrits, %cached)**, use the Profile option of the Status menu, as shown in Figure 33-10.

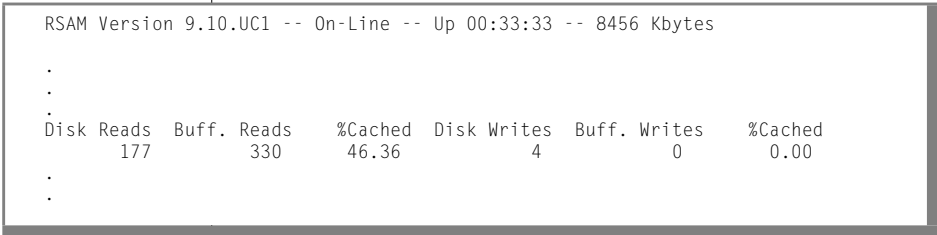


Figure 33-10
Cached Read and Write Statistics in the Profile Option of the ON-Monitor Status Menu

Using SMI Tables

Query the **sysprofile** table to obtain statistics on cached reads and writes and total buffer waits. The following rows are relevant:

- dskreads** number of reads from disk
- bufreads** number of reads from buffers
- dskwrites** number of writes to disk
- bufwrits** number of writes to buffers
- buffwts** number of times 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 Universal 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 Universal Server buffers, see the [INFORMIX-Universal Server Performance Guide](#).

Information on the buffers in each LRU queue consists of the length of the queue and the percentage of the buffers in the queue that have been modified.

Using Command-Line Utilities

You can use the 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 Universal Server attempted to exceed the maximum number of shared buffers specified by the BUFFERS parameter in the ONCONFIG file.

Figure 33-11 shows **onstat -p** output, including the `ovbuff` field.

Figure 33-11
onstat -p Output
Showing *ovbuff*
Field

```
RSAM Version 9.10.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
.
.
.
```

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 12-50](#).) An example of the output is shown in [Figure 33-12 on page 33-26](#). This information tells you when and how the buffers are flushed.

Totals for the following write types are displayed:

- Foreground write
- LRU write
- Chunk write

In addition, **onstat -F** 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 33-12. For more information on the **onstat -F** fields, refer to “[-F Option](#)” on page 39-87.

```
RSAM Version 9.10.UC1 -- On-Line -- Up 01:43:35 -- 8920 Kbytes

Fg Writes      LRU Writes      Chunk Writes
0              146              140

address  flusher  state  data      = 0X0
8067c4   0        I        0
      states: Exit Idle Chunk Lru
```

Figure 33-12
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, refer to “[-R Option](#)” on page 39-99.) Figure 33-13 shows an example of **onstat -R** output.

```
RSAM Version 9.10.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 buufs dirty, stop at 50%
```

Figure 33-13
onstat -R Output

Using SMI Tables

Query the **sysprofile** table to obtain the statistics on write types that are held in the following rows:

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 Universal Server is logging most of the transactions. (The administrator cannot configure or tune the number of latches; Universal Server sets this function internally.) [Figure 33-14 on page 33-28](#) shows **onstat -p** output, including the **lchreqs** and **lchwaits** fields.

```
RSAM Version 9.10.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 33-14
*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 33-15.) 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, Universal Server immediately initiates an abort.*

```
RSAM Version 9.10.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 33-15
onstat -s Output

Using ON-Monitor

To access the **latch Waits** field, mentioned on [page 33-27](#) for **onstat -p**, use the Profile option of the Status menu.

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:

latchreqs	the number of requests for a latch
latchwts	the number of times 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 Universal 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 33-16 shows lock statistics.

Figure 33-16
onstat -p Output
Showing Lock
Statistics

```
RSAM Version 9.10.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

.
.
.
```

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, refer to [page 39-92](#).) An example of the output displayed by this option is shown in Figure 33-17.

Figure 33-17
onstat -k Output

```
RSAM Version 9.10.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
```

Using ON-Monitor

To monitor the same three lock statistics that are mentioned on [page 33-29](#) for the **onstat -p** option, use the Profile option of the Status menu.

Using SMI Tables

Query the **sysprofile** table to obtain statistics on lock use. The following rows contain the relevant statistics:

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:

dbname	database on which the lock is held
tablename	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	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:

npages	the pages allocated to the tblspace
nused	the pages used from this allocated pool
nextns	the number of extents used
npdata	the 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, Universal Server allocates the extent here; if not, Universal Server looks for space in other available chunks. If none of the chunks contains adequate contiguous space, Universal 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 33-18.

```
RSAM Version 9.10.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 33-18
onstat -t Output

Monitoring Virtual Processors

Monitor the virtual processors to determine if the number of virtual processors configured for Universal 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 33-19.

Figure 33-19*onstat -g glo Output*

RSAM Version 9.10.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

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 33-20.

Figure 33-20*onstat -g ioq Output*

RSAM Version 9.10.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

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 33-21 displays **onstat -g rea** output.

```
RSAM Version 9.10.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 33-21
onstat -g rea Output

Using SMI Tables

Query the **sysvppprof** table to obtain information on the virtual processors that are currently running. This table contains the following columns:

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.

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 33-22 on page 33-36](#).

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, ON-Monitor thread, daemon thread and so on). For information on these flags, refer to [“-u Option” on page 39-102](#).
- 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 Universal Server

If you execute **onstat -u** while Universal Server is performing fast recovery, several server threads might appear in the display.

```

RSAM Version 9.10.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 33-22
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 33-23, four scan threads that belong to a decision-support thread are displayed.

```
RSAM Version 9.10.UC1 -- On-Line -- Up 00:50:15 -- 8896 Kbytes
```

```
Threads:
```

tid	tcb	rstcb	prty	status	vp-class	name
.						
.						
11	994060	0	4	sleeping(Forever)	lcpu	kaio
12	994394	80f2a4	2	sleeping(secs: 51)	lcpu	btclean
26	99b11c	80f630	4	ready	lcpu	onmode_mon
32	a9a294	812b64	2	ready	lcpu	sqlexec
113	b72a7c	810b78	2	ready	lcpu	sqlexec
114	b86c8c	81244c	2	cond wait(netnorm)	lcpu	sqlexec
115	b98a7c	812ef0	2	cond wait(netnorm)	lcpu	sqlexec
116	bb4a24	80fd48	2	cond wait(netnorm)	lcpu	sqlexec
117	bc6a24	81161c	2	cond wait(netnorm)	lcpu	sqlexec
118	bd8a24	811290	2	ready	lcpu	sqlexec
119	beae88	810f04	2	cond wait(await_MC1)	lcpu	scan_1.0
120	a8ab48	8127d8	2	ready	lcpu	scan_2.0
121	a96850	810460	2	ready	lcpu	scan_2.1
122	ab6f30	8119a8	2	running	lcpu	scan_2.2

Figure 33-23
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 33-24, session number 49 is running five threads for a decision-support query.

```
RSAM Version 9.10.UC1 -- On-Line -- Up 01:04:36 -- 8896 Kbytes
```

session	id	user	tty	pid	hostname	#RSAM threads	total memory	used 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

Figure 33-24
onstat -g ses Output

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 37-70](#) and the description of **INFORMIXSTACKSIZE** in the *Informix Guide to SQL: Reference*.

Using ON-Monitor

Select the Status menu, User option. The display (shown in Figure 33-25) 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, ON-Monitor thread, daemon thread, and so on)

USER THREAD INFORMATION					
Session	User	Locks Held	Disk Reads	Disk Writes	User thread 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 33-25
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:

sid	the session ID
username	the username (login ID) of the user
uid	the user ID
pid	the process ID
connected	the time that the session started
feprogram	the front-end program (the application that is running as the client)

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. For a full list of the **sysessions** columns, see [“sysessions” on page 38-26](#).



Important: The information in the **sysessions** table is organized by session, while the information displayed by **onstat -u** is organized by thread. Also, unlike the **onstat -u** option, the **sysessions** table does not include information on daemon threads, only user threads.

Query the **sysesprof** 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 [“sysesprof” on page 38-24](#).

Monitoring Parallel Database Query

Monitor the resources (shared memory and threads) that the memory grant manager (MGM) has allocated for parallel database queries (PDQ), and the resources currently used by those queries.

Using Command-Line Utilities

You can use the **onstat -g mgm** option to monitor how MGM coordinates memory use and scan threads. The **onstat** utility reads shared-memory structures and provides statistics that are accurate at the instant that the command executes. Example output is shown in Figure 33-26.

The **onstat -g mgm** display uses a unit of memory called a *quantum*. The *memory quantum* represents a unit of memory, as follows:

$$\text{memory quantum} = \text{DS\_TOTAL\_MEMORY} / \text{DS\_MAX\_QUERIES}$$

The *scan thread quantum* is always equal to 1.

Figure 33-26
onstat -g mgm
 Output

```

RSAM Version 9.10.UC1 -- On-Line -- Up 00:19:05 -- 8896 Kbytes

Memory Grant Manager (MGM)
-----

MAX_PDQPRIORITY: 100
DS_MAX_QUERIES: 5
DS_MAX_SCANS: 10
DS_TOTAL_MEMORY: 4000 KB

Queries:   Active   Ready   Maximum
           3         0         5

Memory:    Total    Free    Quantum
(KB)       4000    3872     800

Scans:     Total    Free    Quantum
           10      8         1

Load Control:   (Memory)   (Scans)   (Priority)   (Max Queries)   (Reinit)
                Gate 1     Gate 2     Gate 3       Gate 4         Gate 5
(Queue Length) 0         0         0           0             0

Active Queries:
-----
Session  Query  Priority  Thread  Memory  Scans   Gate
   7     a3d0c0    1     a8adcc  0/0     1/1    -
   7     a56eb0    1     ae6800  0/0     1/1    -
   9     a751d4    0     96b1b8 16/16    0/0    -

Ready Queries: None

Free Resource      Average #      Minimum #
-----
Memory             489.2 +- 28.7      400
Scans              8.5 +- 0.5       8

Queries            Average #      Maximum #      Total #
-----
Active             1.7 +- 0.7         3             23
Ready             0.0 +- 0.0         0             0

Resource/Lock Cycle Prevention count: 0

```

The first portion of the display shows the values of the PDQ configuration parameters.

The second portion of the display describes MGM internal control information. It is divided into four groups.

The first group is indicated by **Queries**:

Active	Number of PDQ queries that are currently executing
Ready	Number of user queries ready to run, but whose execution Universal Server deferred for load-control reasons
Maximum	Maximum number of queries that Universal Server permits to be active. Reflects current value of the DS_MAX_QUERIES configuration parameter

The next group is indicated by **Memory**:

Total	Kilobytes of memory available for use by PDQ queries (DS_TOTAL_MEMORY specifies this value.)
Free	Kilobytes of memory for PDQ queries not currently in use
Quantum	Kilobytes of memory in a memory quantum

The next group is indicated by **Scans**:

Total	The total number of scan threads as specified by the DS_MAX_SCANS configuration parameter
Free	Number of scan threads currently available for decision-support queries
Quantum	The number of scan threads in a scan-thread quantum

The last group in this portion of the display describes MGM **Load Control**:

Memory	Number of queries that are waiting for memory
Scans	Number of queries that are waiting for scans
Priority	Number of queries that are waiting for queries with higher PDQ priority to run
Max Queries	Number of queries that are waiting for a query slot
Reinit	Number of queries that are waiting for running queries to complete after an onmode -M or -Q

The next portion of the display (**Active Queries**) describes the MGM active and ready queues. This portion of the display shows the number of queries waiting at each gate:

Session	The session ID for the session that initiated the query
Query	Address of the internal control block associated with the query
Priority	PDQ priority assigned to the query
Thread	Thread that registered the query with MGM
Memory	Memory currently granted to the query / memory reserved for the query (Unit is MGM pages (8 kilobytes).)
Scans	Number of scan threads currently used by the query / number of scan threads allocated to the query
Gate	Gate number at which query is waiting

The next portion of the display (**Free Resource**) provides statistics concerning MGM free resources. The numbers in this portion and in the final portion reflect statistics since system initialization or the last **onmode -Q, -M,** or **-S**:

Average	Average of memory / scans
Minimum	Available memory / scans

The last portion of the display (**Queries**) provides statistics concerning MGM queries:

Average	Average active / ready queue length
Minimum	Minimum active / ready queue length
Total	Total active / ready queue length

In addition to the **onstat -g mgm** option, you can use other **onstat** options to obtain information on decision-support queries.

Each decision-support query has a primary thread. This primary thread can start additional threads to perform tasks for the query (for example, scans and sorts). To obtain information on all of the threads that are running for a decision-support query, use the **onstat -u** and **onstat -g ath** options. The **onstat -u** option lists all the threads for a session. If a session is running a decision-support query, the primary thread and any additional threads are listed. For example, session 10 in [Figure 33-22 on page 33-36](#) has a total of five threads running. The **onstat -g ath** option display also lists these threads and includes a **name** column that indicates the role of the thread. For example, [Figure 33-23 on page 33-37](#) lists four *scan* threads, started by a primary (sqlxec) thread.

Use the **onstat -g ses** option to obtain the following information:

- The shared memory allocated for a session that is running a decision-support query
- The shared memory used by a session that is running a decision-support query
- The number of threads that Universal Server started for a session

For example, [Figure 33-24 on page 33-37](#) shows a session (session 49) that has started five threads.

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 Universal 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 33-27.

```
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
```

Figure 33-27
onstat -x Output

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 33-28. For more information on this option, refer to “-g Monitoring Options” on page 39-88.

```
RSAM Version 9.10.UC1 -- On-Line -- Up 07:00:24 -- 8920 Kbytes

Sess  SQL          Current      Iso Lock      SQL  ISAM F.E.
Id    Stmt type    Database      Lv1 Mode     ERR  ERR  Vers
17    ALTER TABLE mydemo      CR  Not Wait   0    0    9.10.UC10
```

Figure 33-28
onstat -g sql Output

Monitoring Databases

Monitor the databases managed by Universal Server to determine the logging status of those databases.

Using ON-Monitor

To find out the logging status of a database from within ON-Monitor, select the Status menu, Databases option. Figure 33-29 shows the output of this option.

Database Name	Owner	In Dbspace	When Created	Log Status
sysmaster	informix	rootdbs	07/28/95	U
mydemo	chrisw	rootdbs	09:08:10	U

Figure 33-29
*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. Universal Server uses the following characters to represent database-logging status:

- 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 Universal 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 38-14](#).

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 Universal 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 Universal 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 9.10.UC1 -- Universal 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, refer to “-l Option” on page 39-93. Figure 33-30 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 33-30
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 Universal 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 33-31. For more information, refer to [“Using -pr Option to Display Reserved-Page Information” on page 39-23](#).

```
.
.
.
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 33-31
oncheck -pr Output
Containing Logical-
Log File Information

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 33-32.

```
.
.
.
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 33-32
*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:

number	the identification number of the logical-log file
uniqid	the unique ID of the log file
size	the size of the file in pages
used	the 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 Universal 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 33-33.

```
RSAM Version 9.10.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 33-33
onstat -l Output
Showing Physical-
Log Information

oncheck -pr

Execute **oncheck -pr** to obtain the physical-log file information that Universal 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 33-34.

```
Validating INFORMIX-Universal 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 33-34
oncheck -pr Output
That Includes
Physical-Log
Information

Using ON-Monitor

All the information on the physical-log file that the **onstat -l** utility provides is also available from the Logs option of the Status menu.

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, refer to the [INFORMIX-Universal Server 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 33-35.

```
RSAM Version 9.10.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 33-35
onstat -l Output
Showing Log-
Buffer Information

Using ON-Monitor

All the information that the **onstat -l** utility provides on the physical-log and logical-log buffers is also available from the Logs option of the Status menu. An example of the output is shown in Figure 33-36.

```
PHYSICAL LOG:
Buffer  Bufsize  Bufused  Numpages  Numwrits  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  Numwrits  Recs/Page  Pages/IO
L-2     16       0         1         1         1         1.00     1.00
```

Figure 33-36
*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:

- plgpagewrites** the number of pages written to the physical-log buffer
- plgwrites** the number of writes from the physical-log buffer to the physical log file

llgreys	the number of records written to the logical-log buffer
llgpagewrites	the number of pages written to the logical-log buffer
llgwrites	the number of writes from the logical-log buffer to the logical-log files

Monitoring Chunk Status

Monitor chunk status to determine the following information:

- Whether dbspaces and blobspaces are mirrored
- Whether a chunk within a mirrored dbspace is the primary chunk or the mirrored chunk
- Whether a chunk is on-line, down, or being recovered

Using Command-Line Utilities

You can use the following command-line options to obtain information about chunk status.

onstat -d

The **onstat -d** utility lists all the dbspaces and blobspaces and the chunks within those spaces. Example output is shown in [Figure 33-37 on page 33-56](#). 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

For descriptions of the **onstat -d** flags, refer to “-d Option” on page 39-84.

Figure 33-37
onstat -d Output

```
RSAM Version 9.10.UC1 -- Quiescent -- Up 00:04:05 -- 4584 Kbytes

Dbspaces
address  number  flags  fchunk  nchunks  flags  owner  name
40d100   1          1      1        1        N      informix rootdb
40d144   2          2      2        1        M      informix fstspace
40d188   3         10      3        1        N B    informix fstblob
  3 active, 10 total

Chunks
address  chk/dbs  offset  size    free    bpages  flags  pathname
40c274   1        1      0    20000  13417      PO-   /home/server/root_chunk
40c30c   2        2      0     2000   1643      PO-   /home/server/test_chunk
40c8fc   2        2      0     2000     0      MO-   /home/server/test_mirr
40c3a4   3        3      0     4000   ~998    1000  POB    /home/server/blob_chunk
  3 active, 10 total
```

oncheck -pr

The **oncheck -pr** utility displays the dbspace and chunk-mirror status information that is stored in the reserved pages PAGE_1DBSP, PAGE_2DBSP, PAGE1_PCHUNK, and PAGE2_PCHUNK.

Pages PAGE_1DBSP and PAGE_2DBSP contain flags to indicate if the dbspace is mirrored and if the dbspace is a blobspace. For an explanation of the flag values, refer to “PAGE_DBSP” on page 42-11.

Pages PAGE_1PCHUNK and PAGE_2PCHUNK contain flags to indicate whether each chunk is a mirrored chunk and whether the chunk is on-line or down. For an explanation of the flag values, refer to “PAGE_PCHUNK” on page 42-11.

These **oncheck -pr** flags contain the same information as the **onstat -d** flags, except that they do not reflect any changes since the last checkpoint.

Using ON-Monitor

Select the Status menu, Spaces option. The first screen indicates whether the space is mirrored. An example of this output is shown in Figure 33-38.

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/BLOBSACES

Id	Name	Number of Chunks	When Created	Status
1	rootdbs	1	08/05/95	N
2	fstspace	1	08/09/95	Y
3	fstblob	1	08/09/95	N

Figure 33-38
First Screen of
ON-Monitor Spaces
Option

The second screen indicates the mirror status of each chunk. An example is shown in Figure 33-39. These flags correspond to the flags displayed by **onstat -d**. For an explanation of the flag values, refer to “[-d Option](#)” on [page 39-84](#).

Press ESC to return to the Status Menu.
Use arrow keys to move the cursor.

CHUNKS FOR cooked space

Chunk Id	Chunk Offset	Pages In Chunk	Pages Used	Full Pathname of Chunk	Status
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 33-39
Second Screen of
ON-Monitor Spaces
Option

Using SMI Tables

Query the **sysdbspaces** table to determine if a dbspace is mirrored. The following columns are relevant:

dbnum	the dbspace number
name	the name of the dbspace
is_mirrored	whether the dbspace is mirrored

Query the **syschunks** table to obtain the status of a chunk. The following columns are relevant:

chknun	the number of the chunk within the dbspace
dbnum	the number of the dbspace
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

Monitoring the Database Server for Disabling I/O Errors

Universal 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

Universal 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 Universal Server entity  
Action: What action the Universal 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	Dbospace/blobspace {space-name} is disabled.	Restore dbospace/blobspace {space-name}.
ABORT	Universal Server must abort.	Reinitialize shared memory.
WAIT	Universal 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 Universal Server sends to the message log, see [Chapter 40, “Message-Log Messages.”](#)

Using Event Alarms to Monitor Disabling I/O Errors

When a dbospace incurs a disabling I/O error, Universal Server passes the following values as parameters to your event-alarm executable file.

Parameter	Value
Severity:	4 (Emergency)
Class:	5
Class message:	dbospace is disabled: 'dbospace-name'
Specific message:	Chunk {chunk-number} is being taken OFFLINE.

If you want Universal Server to use event alarms to notify you about disabling I/O errors, you must write a script that Universal Server executes when it detects a disabling I/O error. For information about how to set up this executable file and make Universal Server aware of the location of the executable file that you write, see [“Sources of Information for Monitoring the Database Server” on page 33-6.](#)

Monitoring Disk Usage

This section describes methods of tracking the disk space used by various Universal Server storage units.

For background information about internal Universal Server storage units mentioned in this section, refer to [Chapter 42, “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.

onstat -d

The following information is displayed for each chunk within a dbspace (or blobspace):

- 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

For more information on this option, refer to “-d Option” on page 39-84. An example of **onstat -d** output is provided in Figure 33-37 on page 33-56.

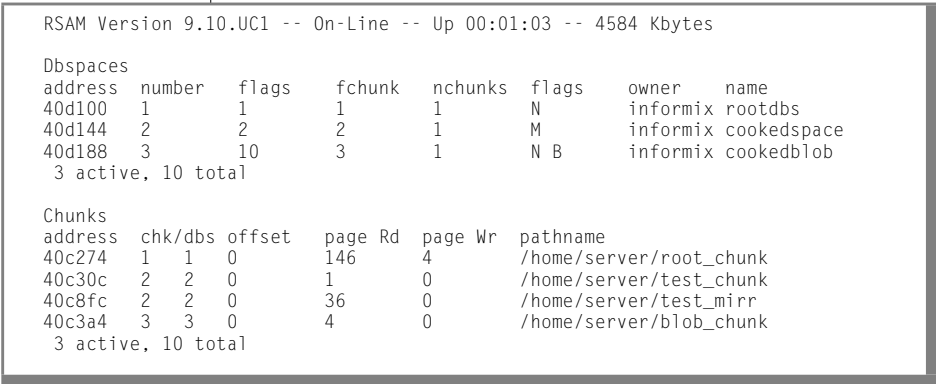
onstat -D

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 33-40.

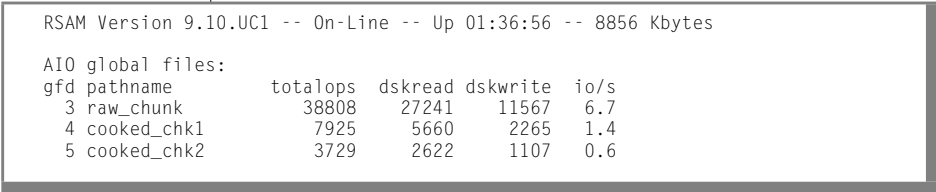
Figure 33-40
onstat -D Output



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 33-41.

Figure 33-41
onstat -g iof Output



oncheck -pr

Execute **oncheck -pr** to obtain the chunk information that Universal 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 33-42.

```
Validating INFORMIX-Universal 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-Universal 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 33-42
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 Universal 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 33-43.

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 DSpace 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 33-43
oncheck -pe Output

Using ON-Monitor

The Spaces option of the Status menu displays the same information as the **onstat -d** option. Example output is shown in [Figure 33-38 on page 33-57](#) and [Figure 33-39 on page 33-57](#).

Using SMI Tables

Query the **syschunks** table to obtain the chunk size and the number of free pages. The following columns are relevant:

chknum	the number of the chunk within the dbspace
dbsnun	the number of the dbspace
chksize	the total size of the chunk in pages
nfree	the number of pages that are free

The **syschkio** table contains the following columns:

pagesread	the number of pages read from the chunk
pageswritten	the number of pages written to the chunk

Monitoring Tblspaces and Extents

Monitor tablespaces 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 tablespaces and extents.

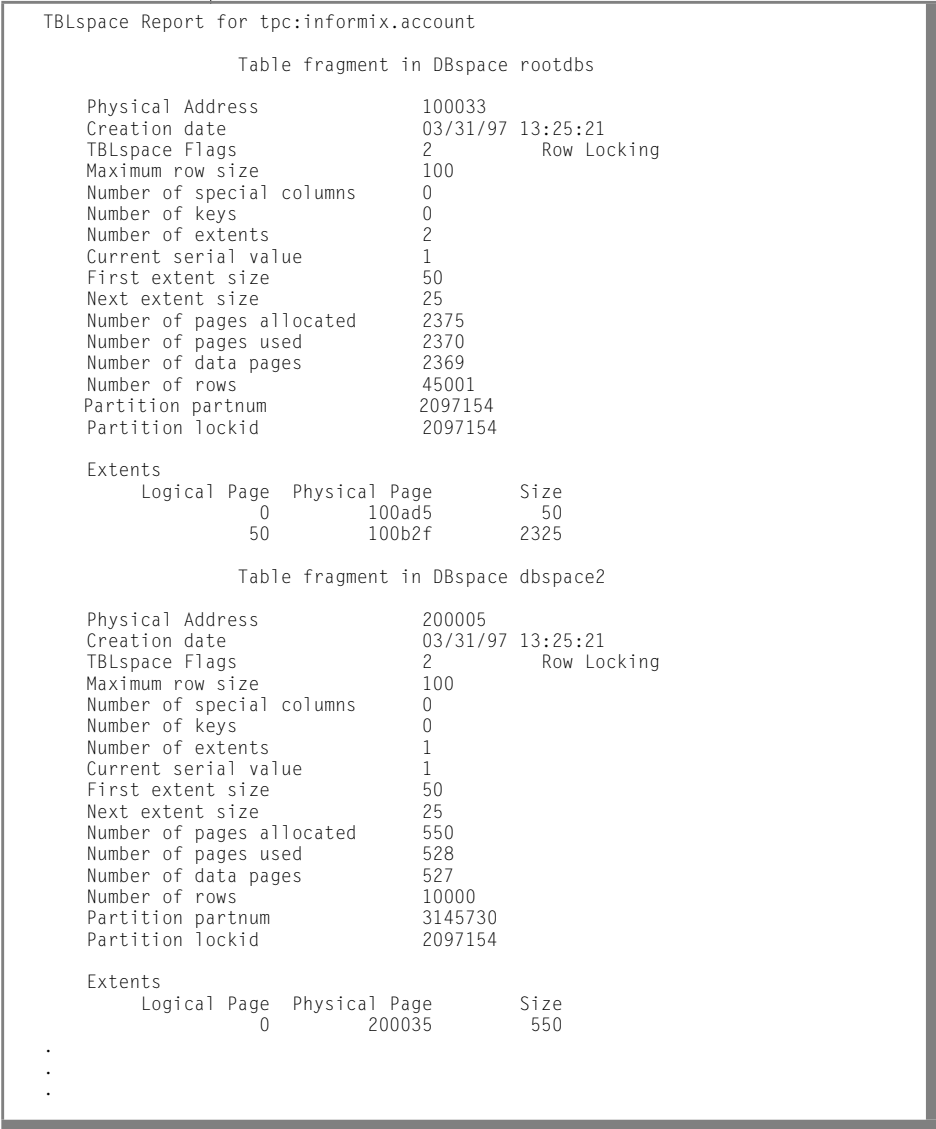
oncheck -pt

Execute **oncheck -pt** with a database-name or table-name parameter to obtain the following information for each tablespace 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 33-44. 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.

Figure 33-44
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 33-45. 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					
Level	Total	Average No. Keys	Average Free Bytes		

1	1	4	1973		
2	4	116	506		
3	466	128	217		

Total	471	128	223		

Figure 33-45
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:

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 Sbspaces

Monitor sbspaces 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 smart large objects in sbspaces.

oncheck -cs and oncheck -cS

The **oncheck -cs** and the **oncheck -cS** options validate the metadata area of an sbspace. [Figure 33-46 on page 33-68](#) shows an example of the output from the **-cs** option for **sbspace1**. If you do not specify an sbspace name on the command line, **oncheck** checks and displays the metadata for all sbspaces.

Validating space 'sbspacel' ...

SBL0Bspace Partition	Partnum	Used	Free
sbspacel:'informix'.TBLSpace	0x200001	6	44
sbspacel:'informix'.sbspacel_desc	0x200002	2	2
sbspacel:'informix'.chunk_adjunc	0x200003	2	2
sbspacel:'informix'.L0_hdr_partn	0x200004	2	6
sbspacel:'informix'.L0_ud_free	0x200005	2	2

Figure 33-46
oncheck -cs Output

The **-cs** option checks smart-large-object extents in addition to the sbspacel partitions.

oncheck -ps and oncheck -pS

The **oncheck -ps** and **oncheck -pS** options validate and display information about the metadata areas in sbspacel partitions. Figure 33-47 shows an example of the output from the **-ps** option for **sbspacel1**. If you do not specify an sbspacel name on the command line, **oncheck** validates and displays information for all sbspaces.

Validating space 'sbspacel' ...

```
TBLSpace Report for
  TBLspace Flags          2801      Page Locking
                                TBLspace use 4 bit bit-maps
                                Permanent System TBLspace

Partition partnum        2097153
Number of rows           84
Number of special columns 0
Number of keys           0
Number of extents        1
Current serial value      1
First extent size        50
Next extent size         50
Number of pages allocated 50
Number of pages used      6
Number of data pages      0
Number of rows           0
Partition lockid         2097153
Optical Cluster Partnum   -1
Current SERIAL8 value     1
Current REFID value       1
Created                  Sat Nov  9 20:03:48 1996
```

Figure 33-47
oncheck -ps Output

The **-pS** option displays information about smart-large-object extents in addition to the information about metadata areas in sbspace partitions.

Monitoring Simple Large Objects 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 simple large objects in a blobspace.

onstat -d

The **onstat -d** option displays information on each blobspace and the chunks within each blobspace. Figure 33-48 shows an example of the output from this option. In this example, a blobspace called **fstblob** contains a single chunk called **blob_chunk**.

Figure 33-48
onstat -d Output
with Information
About Simple Large
Objects

```
RSAM Version 9.10.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   P0-    /home/server/root_chunk
40c2bc   2        2      0       2000   1659    P0-    /home/server/fst_chunk
40c354   3        3      0       4000   ~924    P0B    /home/server/blob_chunk
3 active, 10 total
```

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 simple large object deletion occurred is backed up, and the blobpage is freed. Therefore, if you delete 25 simple large objects and immediately execute **onstat -d**, the newly freed space does not appear in the **onstat** output.

onstat -O

The **onstat -O** option displays information about the staging-area blobspace and the INFORMIX-OnLine/Optical memory cache. Figure 33-49 shows an example of the output from this option. The totals shown in the display accumulate from session to session. Universal Server resets the totals to 0 only when you execute **onstat -z**.

```
INFORMIX-Universal Server 9.10.UC1 -- On-Line -- Up 00:56:39 -- 10864 Kbytes

Subsystem not available

Optical StageBlob Cache
System Cache Totals:
Size  Alloc. Avail.      Number Kbytes      System Blob Totals:
0      0      0          0      0          Number Kbytes
0      0          0      0      0

User Cache Totals:
SID    User      Size Number Kbytes      User Blob Totals:
No sessions currently using the cache          Number Kbytes
```

Figure 33-49
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 Universal Server allocated to the cache.
avail	describes how much of alloc (in kilobytes) is not used.
number	is the number of simple large objects that Universal Server successfully put into the cache without overflowing.

kbytes	is the number of kilobytes of the simple large objects that Universal Server put into the cache without overflowing.
number	is the number of simple large objects that Universal Server wrote to the staging-area blobSpace.
kbytes	is the number of kilobytes of the simple large objects that Universal Server wrote to the staging-area blobSpace.

Although the **size** output indicates the amount of memory that is specified in the configuration parameter **OPCACHEMAX**, Universal Server does not allocate memory to **OPCACHEMAX** until necessary. Therefore, the **alloc** output reflects only the number of 1-kilobyte pieces of the largest simple large object that has been processed. When the values in the **alloc** and **avail** output are equal, the cache is empty.

The second section of the display describes the following user-cache totals information:

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, Universal Server uses the size that you specify in the configuration parameter OPCACHEMAX .
number	is the number of simple large objects that Universal Server put into cache without overflowing.
kbytes	is the number of kilobytes of the simple large objects that Universal Server put into the cache without overflowing.
number	is the number of simple large objects that Universal Server wrote to the staging-area blobSpace.
kbytes	is the size of the simple large objects (in kilobytes) that Universal Server wrote to the staging-area blobSpace.

oncheck -pB

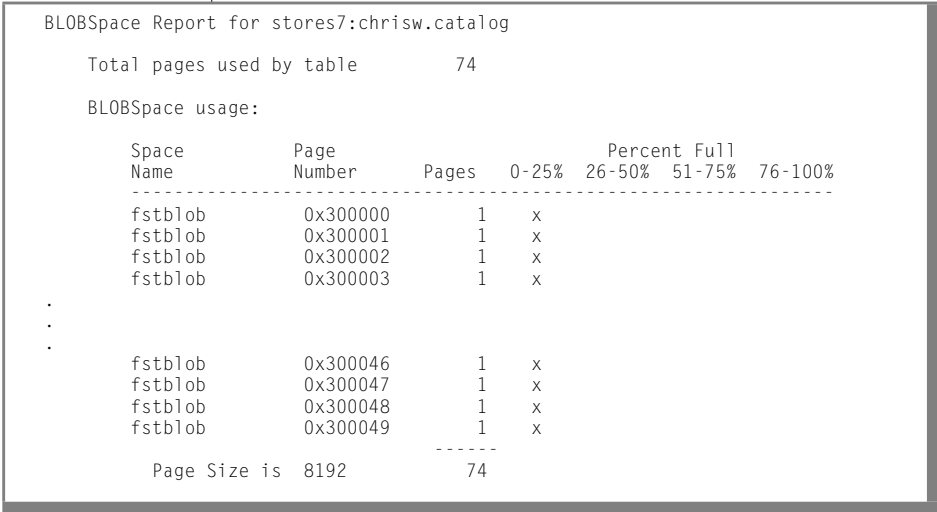
This option gathers its data from the actual simple-large-object storage statistics.

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 simple large object, for each simple large object 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 33-50.

Figure 33-50
oncheck -pB Output



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 simple large objects, 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 Universal Server pages, *not* blobpages.

Example output is shown in Figure 33-51.



BLOBSpace Usage Report: fstblob		Owner: informix	Created: 03/01/97
Chunk: 3	/home/server/blob_chunk	Size	Used
		4000	304
			Free
			3696
Disk usage for Chunk 3		Total Pages	
-----		-----	
OVERHEAD		8	
stores7:chrisw.catalog		296	
FREE		3696	

Figure 33-51
oncheck -pe Output
Showing BlobSpace
Use

Using ON-Monitor

When you select the Spaces option of the Status menu, Universal 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 33-39 on page 33-57](#).

Monitoring Simple Large Objects in a Dbspace

You can monitor dbspaces to determine the number of dbspace pages that are used by simple large objects.

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, Universal 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 simple large object information. Example output is shown in [Figure 33-52 on page 33-75](#).

Universal Server can store more than one simple large object on the same dbspace blobpage. Therefore, you can count the number of pages that store simple large objects in the tblspace, but you cannot estimate the number of simple large 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

Level	Total	Average No. Keys	Average Free Bytes
1	1	74	1058
Total	1	74	1058

Index Usage Report for index 111_18 on mydemo:chrisw.catalog

Level	Total	Average No. Keys	Average Free Bytes
1	1	74	984
Total	1	74	984

Figure 33-52
*oncheck -pT Output
 with a TbLspace
 Report That
 Contains Simple
 Large Objects*

Monitoring Data-Replication Status

Monitor the data-replication status of a Universal Server 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 data replication is on
- The values of the 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 Universal Server database server.

The following example shows a header for a database server that is the primary Universal Server database server in a data-replication pair, and in on-line mode:

```
RSAM Version 9.10.UC1      -- Universal Server (Prim) -- Up 45:08:57 -- 21604 Kbytes
```

This example shows a database server that is the secondary database server in a data-replication pair, and in read-only mode.

```
RSAM Version 9.10.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 data replication. The type for this database server is standard.

```
RSAM Version 9.10.UC1      -- Universal Server -- Up 20:10:57 -- 21604 Kbytes
```

onstat -g dri

To obtain full data-replication monitoring information, execute the **onstat -g dri** option. The following fields are displayed:

- The database server type (primary, secondary, or standard)
- The data-replication state (on or off)
- The paired database server

- The last data-replication checkpoint
- The values of the data-replication configuration parameters

Example output is shown in Figure 33-53. This example shows a primary Universal Server database server, paired with a secondary Universal Server database server that has the DBSERVERNAME of **beach_ol**. Data replication has been started.

Figure 33-53
onstat -g dri Output

RSAM Version 9.10.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		

oncheck -pr

If your database server is running data replication, the reserved pages PAGE_1ARCH and PAGE_2ARCH store the checkpoint information that data replication uses to synchronize the primary and secondary database servers. An example of the relevant **oncheck -pr** output is given in Figure 33-54.

Figure 33-54
*oncheck -pr
PAGE_1ARCH
Output for Database
Server Running
Data Replication*

Validating INFORMIX-Universal 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

Using ON-Monitor

Use the Status menu, data-Replication option to see information about 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 38-17](#), contains the following columns:

type	data-replication server type
state	data-replication server state
name	server name
intvl	data-replication buffer flush interval
timeout	network timeout
lostfound	data-replication lost+found pathname

Distributed Data

Section VIII



Multiphase Commit Protocols

Two-Phase Commit Protocol	34-4
When Is the Two-Phase Commit Protocol Used?	34-4
What Goals Does the Two-Phase Commit Protocol Achieve?	34-5
Two-Phase Commit Concepts	34-5
Phases of the Two-Phase Commit Protocol	34-6
Precommit Phase	34-7
Postdecision Phase	34-7
Examples of Two-Phase Commit Transactions	34-8
How the Two-Phase Commit Protocol Handles Failures.	34-10
What Types of Failures Does Automatic Recovery Handle?	34-10
What Is the Administrator's Role in Automatic Recovery?	34-10
Automatic-Recovery Mechanisms for Coordinator Failure	34-10
Automatic-Recovery Mechanisms for Participant Failure	34-14
Presumed-Abort Optimization	34-17
How Does Presumed-Abort Optimization Affect Automatic Recovery?	34-18
Why Is an Optimization Realized?	34-18
Independent Actions	34-18
What Initiates Independent Action?	34-19
Possible Results of Independent Action	34-19
Independent Actions That Allow Successful Completion of Transaction	34-20
Independent Actions That Result in an Error Condition	34-20
Independent Actions That Result in Heuristic Decisions	34-21
The Heuristic Rollback Scenario	34-22
Conditions That Result in a Heuristic Rollback	34-22
What Happens When a Heuristic Rollback Occurs?	34-24

The Heuristic End-Transaction Scenario	34-26
When to Perform a Heuristic End Transaction	34-26
How to Use onmode -Z.	34-28
What Happens When the Transaction Is Ended	
Heuristically?	34-28
Tracking a Global Transaction	34-29
Two-Phase Commit Protocol Errors	34-29
Two-Phase Commit and Logical-Log Records.	34-30
Logical-Log Records When the Transaction Commits.	34-31
Logical-Log Records Written During a Heuristic Rollback	34-33
Logical-Log Records Written After a Heuristic End Transaction	34-35
Configuration Parameters Used in Two-Phase Commits	34-37
Function of the DEADLOCK_TIMEOUT Parameter	34-37
Function of the TXTIMEOUT Parameter	34-37
Heterogeneous Commit Protocol	34-38
Which Gateways Can Participate in a Heterogeneous Commit	
Transaction?	34-39
Enabling and Disabling Heterogeneous Commit	34-40
How Does Heterogeneous Commit Work	34-41
Precommit Phase	34-41
Gateway Commit Phase	34-41
Heterogeneous Commit Optimization.	34-42
Implications of a Failed Heterogeneous Commit	34-43
Database Server Coordinator Failure	34-43
Participant Failure	34-44
Interpreting Heterogeneous Commit Error Messages	34-45

A multiphase commit protocol is a mechanism that INFORMIX-Universal Server uses to process transactions that span multiple Universal Server database servers. Understanding how multiphase commit protocols work becomes important only when a deviation from one of these protocols occurs. Universal Server supports two multiphase commit protocols: two-phase commit protocol and heterogeneous commit protocol.

This chapter covers the following topics:

- The two-phase commit protocol
- Consequences of deviating from the two-phase commit protocol
- Two-phase commit errors
- Logical-log records written during a two-phase commit transaction
- Configuration parameters that are specific to distributed environments
- The heterogeneous commit protocol
- How to enable the heterogeneous commit protocol
- How the heterogeneous commit protocol works
- Implications of a failed heterogeneous commit

For information on recovering manually from a failed two-phase commit, refer to [Chapter 35, “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 Universal Server 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 34-1, which includes three Universal Server database servers.

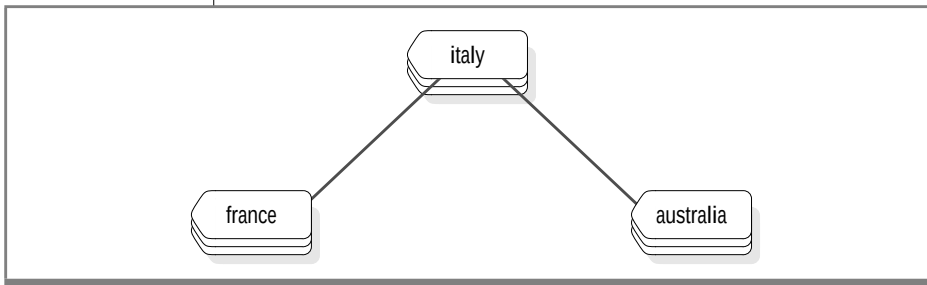


Figure 34-1
*Connected
Database Servers*

If you execute the commands shown in Figure 34-2, 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 34-2
*Example of a
Distributed
Transaction*

Universal 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 Universal Server database servers implement the same action (either a commit or a rollback), regardless of local or network failures during the protocol

If any Universal Server database server is unable to commit its portion of the transaction, *all* Universal Server database servers participating in the transaction must be prevented from committing their work. Universal 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* Universal Server 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 34-2 on page 34-4](#), 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 33-45](#).

In the two-phase commit protocol, the transaction that is under the direction of the coordinator is called the *global transaction*.

- Participants

The Universal Server database servers that perform operations under the direction of the coordinator are the participants. In [Figure 34-1 on page 34-4](#), the participants are **france** and **australia**. The work that each participant Universal Server database server performs is a *piece of work* associated with the global transaction. In this example, the coordinator Universal 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 Universal Server 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 34-30](#).

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 34-3 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 34-3

Two-Phase Commit Protocol That Results in a Committed Transaction

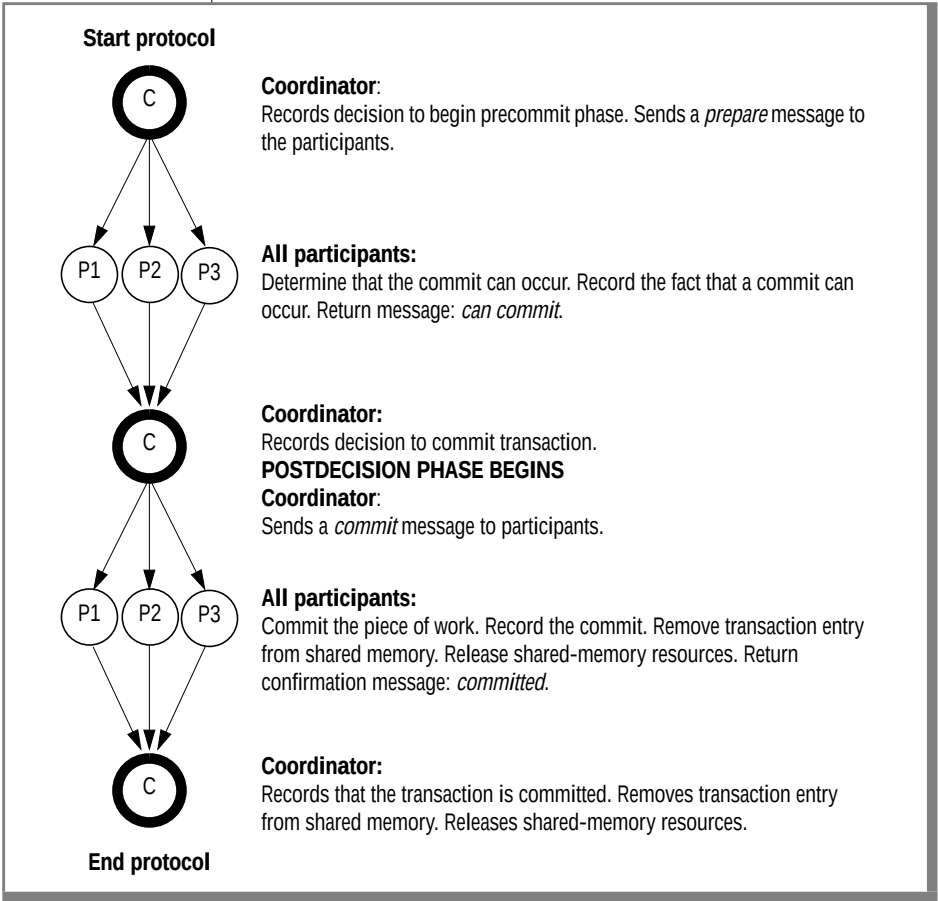
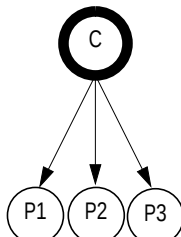


Figure 34-4 represents a two-phase commit that results in a rolled-back transaction. For some reason, probably a deferred constraint violation, the P1 participant determines that the commit cannot occur and returns a `cannot commit` message to the coordinator.

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.

Start protocol



Coordinator:

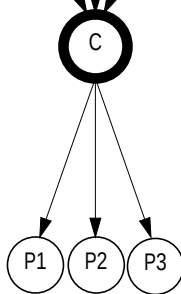
Records decision to begin precommit phase. Sends a *prepare* message to the participants.

Participant 1:

Determines that the commit cannot occur. Returns message: *cannot commit*. Rolls back piece of work.

Participants 2 and 3:

Determine that the commit can occur. Record the fact that a commit can occur. Return message: *can commit*.



Coordinator:

Receives *cannot commit* response. Records decision to roll back transaction.

PHASE 2 BEGINS

Coordinator:

Rolls back work on behalf of transaction. Records rollback. Removes transaction entry from shared memory. Releases shared-memory resources. Sends a *roll back* message to participants.

Participants 2 and 3:

Roll back piece of work. Remove transaction entry from shared-memory transaction table. Release shared-memory resources.

End protocol

Figure 34-4

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 Universal Server 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 Universal 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 34-17.](#))

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). Since 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). Since 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 Universal Server; otherwise, coordinator recovery begins as part of Universal Server processing.

As part of its regular activity, the **main_loop** thread at the coordinator Universal 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 Universal 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.

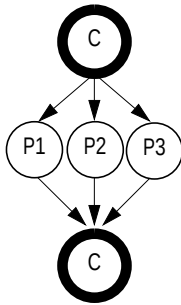
If the new coordinating thread receives a message that the transaction status is `can commit`, it sends a message to the participant to commit the transaction. After all participants send acknowledgments indicating that their piece of work is committed, the coordinating thread ends the transaction.

If the coordinating thread cannot contact one or more of the participants, or if execution errors are detected, messages are recorded in the Universal Server message log.

The coordinating thread then continues its attempts to contact all participants and to complete the protocol. Figure 34-5 illustrates the coordinator-recovery process.

Figure 34-5
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*.

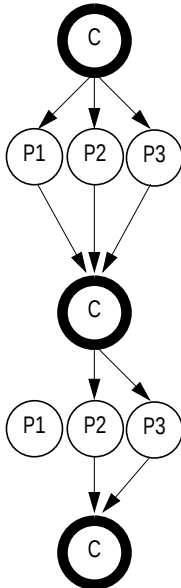
Coordinator:

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

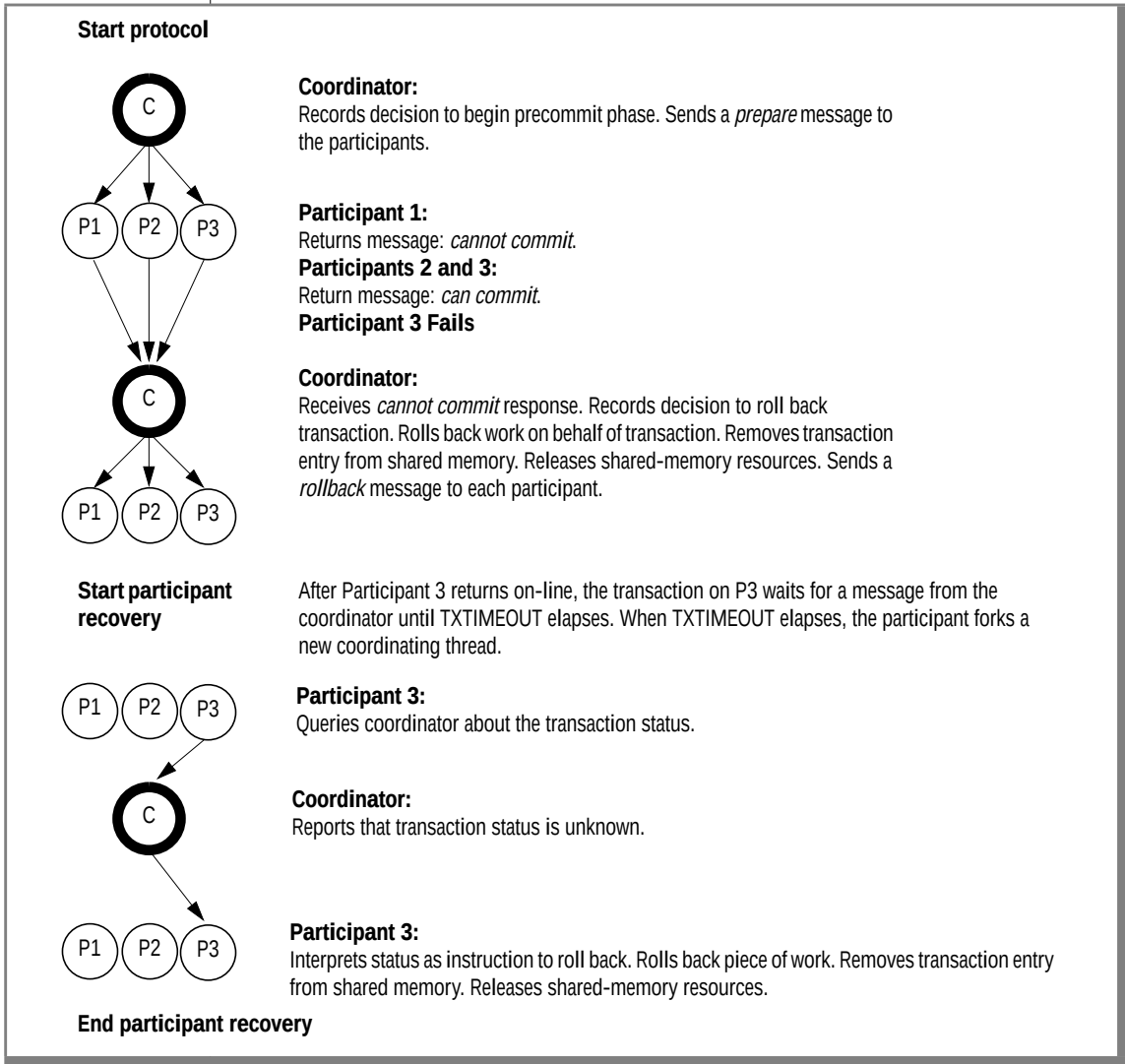
If the participant is down, the coordinator is not able to fork a new participant thread. The coordinator continues to loop through this procedure, however, until it succeeds (when the system administrator brings the participant back on-line).

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. Since 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 34-6 on page 34-16](#) illustrates this scenario.

Figure 34-6
Automatic 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 34-14](#). 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 34-14](#). 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 Universal 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 34-10.](#))

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 35, “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 Universal Server 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 Universal Server 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 Universal Server 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 Universal 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 Universal Server 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 Universal 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 Universal 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 Universal 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 Universal Server 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 34-22](#))
- Heuristic end transaction (described in [“The Heuristic End-Transaction Scenario” on page 34-26](#)).

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 Universal Server 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 37, “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 Universal Server 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, Universal 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, Universal Server has initiated a heuristic rollback. That is, this Universal Server 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 Universal Server database server where a rollback actually closes the transaction.

The logical log might continue to fill until the exclusive high-water mark is reached (LTXEHW). 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 Universal 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 Universal Server where the rollback occurred, a record is placed in the Universal Server logical log (type HEURTX). Locks and resources held by the transaction are freed. The participant thread writes the following message in the Universal 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 Universal Server 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 Universal 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 Universal 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 Universal Server 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 Universal Server database servers that reported a heuristic rollback. If a large number of Universal Server database servers rolled back the transaction, this list could be truncated. The complete list is always included in the message log for the coordinator Universal Server.

In this situation, examine the logical log at each participant Universal Server site and determine whether your database system is consistent. (See [“Determine If a Transaction Was Implemented Inconsistently” on page 35-4.](#))

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 Universal 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 Universal 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 Universal 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 Universal 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 Universal 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, Universal 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 Universal Server Transaction” on page 39-41](#).

The *address* parameter is obtained from **onstat -x** output. See [“-x Option” on page 39-104](#).

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 Universal 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 Universal Server database server, which no longer has information about the transaction. The lack of a transaction-table entry at the participant Universal Server database server indicates that the transaction committed. The coordinator Universal Server 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 example output of the **onstat -x** utility, refer to [“Monitoring Transactions” on page 33-45](#). The fields displayed by **onstat -x** are described in detail in [“-x Option” on page 39-104](#).

Two-Phase Commit Protocol Errors

Three two-phase commit protocol errors require special attention from the administrator:

Error -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 34-24 . For an explanation of how the inconsistent transaction developed and to learn the options available to you, refer to this discussion.
Error -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 34-24 . For an explanation of how the inconsistent transaction developed, refer to this discussion.
Error -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 34-20 .

Two-Phase Commit and Logical-Log Records

Universal 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 35, “Recovering Manually from Failed Two-Phase Commit.”](#))

The following logical-log records are involved in distributed transactions:

- BEGPREP
- PREPARE
- TABLOCKS
- HEURTX
- ENDTRANS

These logical-log records are described in [“Logical-Log Record Types and Additional Columns” on page 41-7.](#)

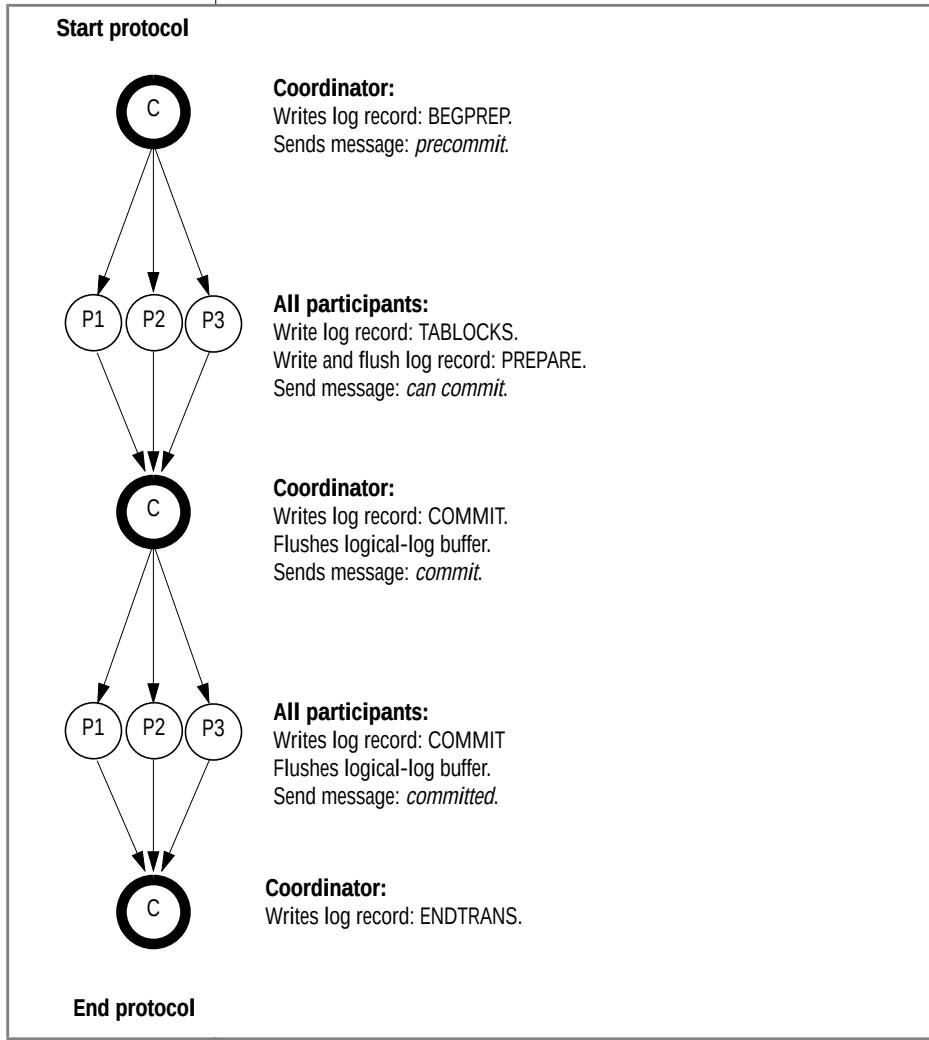
This section examines the sequence of logical-log records that are written during the following Universal Server scenarios:

- 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 34-7 illustrates the writing sequence of the logical-log records during a successful two-phase commit protocol that results in a committed transaction.

Figure 34-7
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.

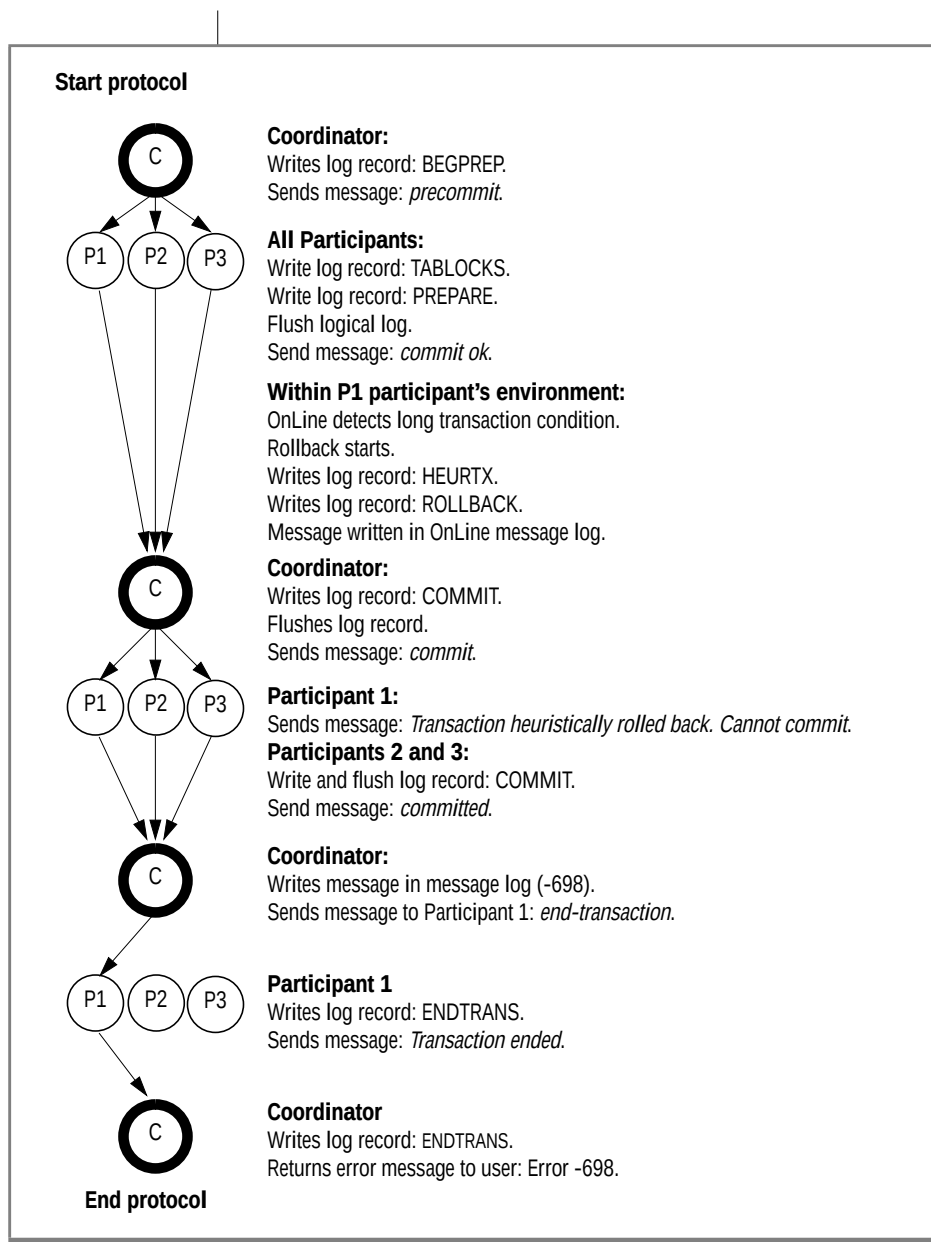
The participants in [Figure 34-7 on page 34-31](#) must immediately flush both the PREPARE and the COMMIT logical-log records. Flushing the PREPARE record ensures that, if the participant's host computer fails, fast recovery is able to determine that this participant is part of a global transaction. 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 34-8 on page 34-34](#) illustrates the sequence in which Universal 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 34-7 on page 34-31](#). 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) Universal Server database server. The end result is a transaction that is inconsistently implemented. See [“The Heuristic Rollback Scenario” on page 34-22](#).

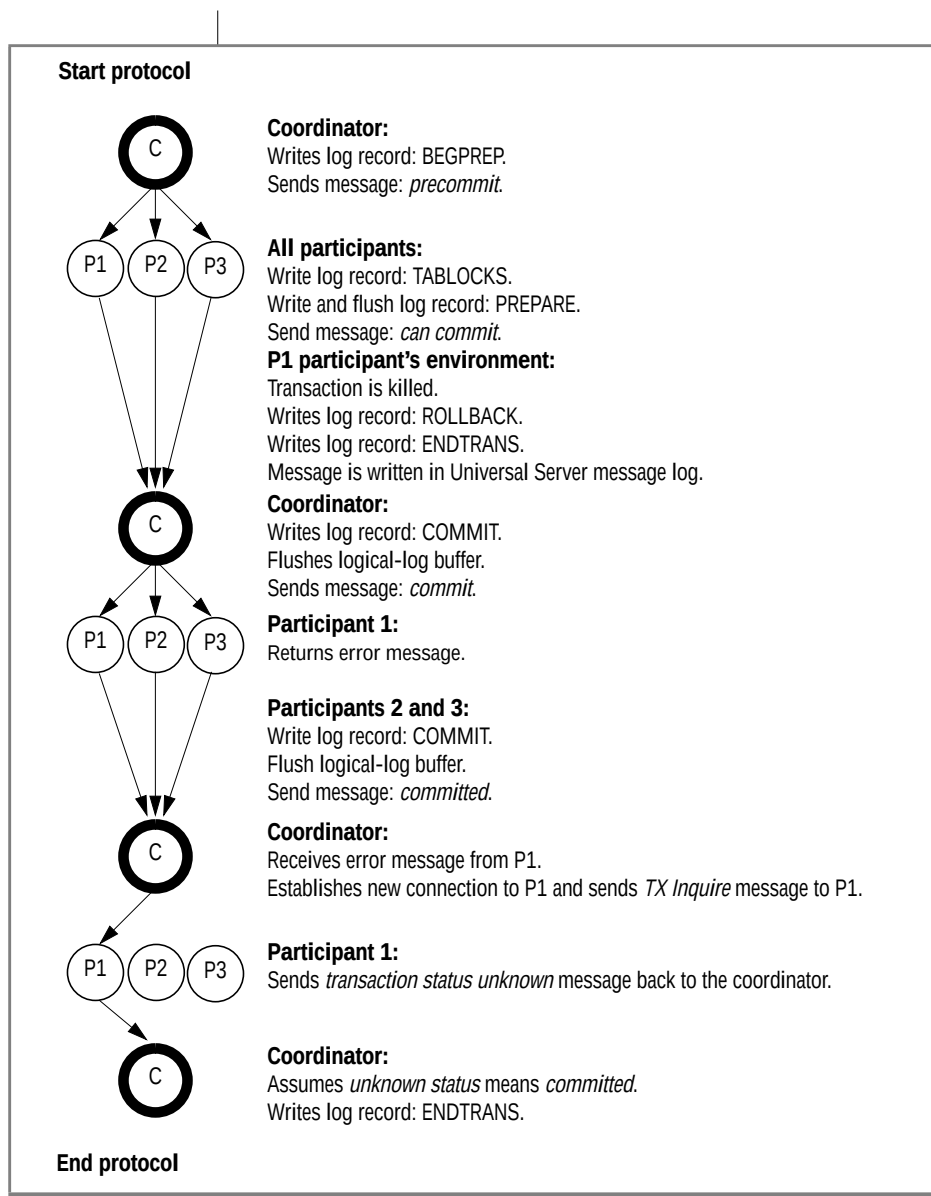
Figure 34-8
Logical-Log
Records Written
During a Heuristic
Rollback



Logical-Log Records Written After a Heuristic End Transaction

Figure 34-9 on page 34-36 illustrates the writing sequence of the logical-log records during a heuristic end transaction. The event is always the result of a Universal Server administrator killing a transaction (see [“Kill a Universal Server Transaction” on page 39-41](#)) at a participant Universal Server database server after the participant has sent a `can commit` message. In Figure 34-9, 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 34-26](#).

Figure 34-9
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` 5-18
- `TXTIMEOUT` 5-73

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 Universal Server 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 Universal Server waits to receive a *commit* instruction from a coordinator Universal Server during a distributed transaction. If the period of time specified by `TXTIMEOUT` elapses, the participant Universal 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 34-10.](#)

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 OnLine Dynamic Server, Version 7.2 or later, or Universal 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.

Which Gateways Can Participate in a Heterogeneous Commit Transaction?

Figure 34-10 illustrates this scenario.

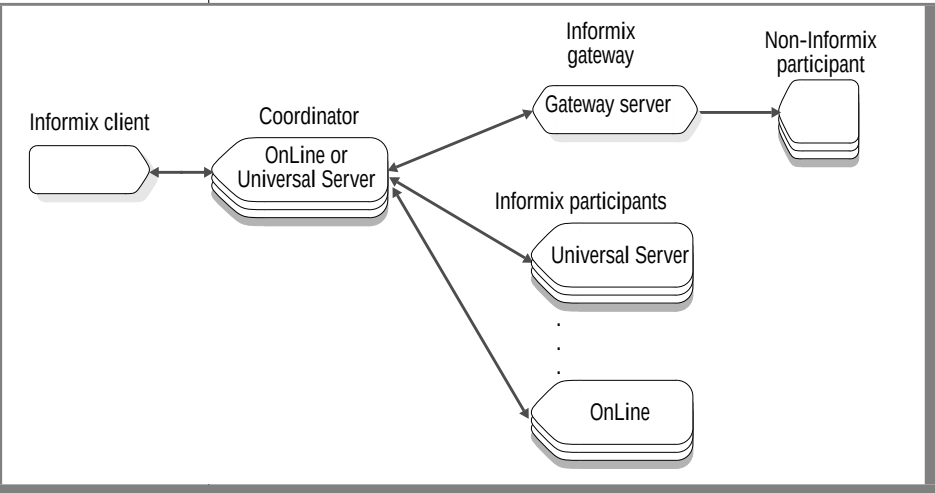


Figure 34-10
*Configuration That
Requires
Heterogeneous
Commit for
Distributed
Transactions*

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 Universal Server uses the heterogeneous commit protocol.

Gateway	Database Servers
INFORMIX-Enterprise Gateway <i>with DRDA</i>	IBM DB2, OS / 400, SQL / DS
INFORMIX-Enterprise Gateway <i>for EDA/SQL</i>	EDA / SQL
INFORMIX-Enterprise Gateway Manager	Any database server with ODBC connectivity

Enabling and Disabling Heterogeneous Commit

You can enable or disable heterogeneous commit in one of two following ways:

- By using ON-Monitor
- By changing 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?	Protocol That Universal Server Uses
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 34-7](#).

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 34-11 on page 34-42](#) illustrates this process.

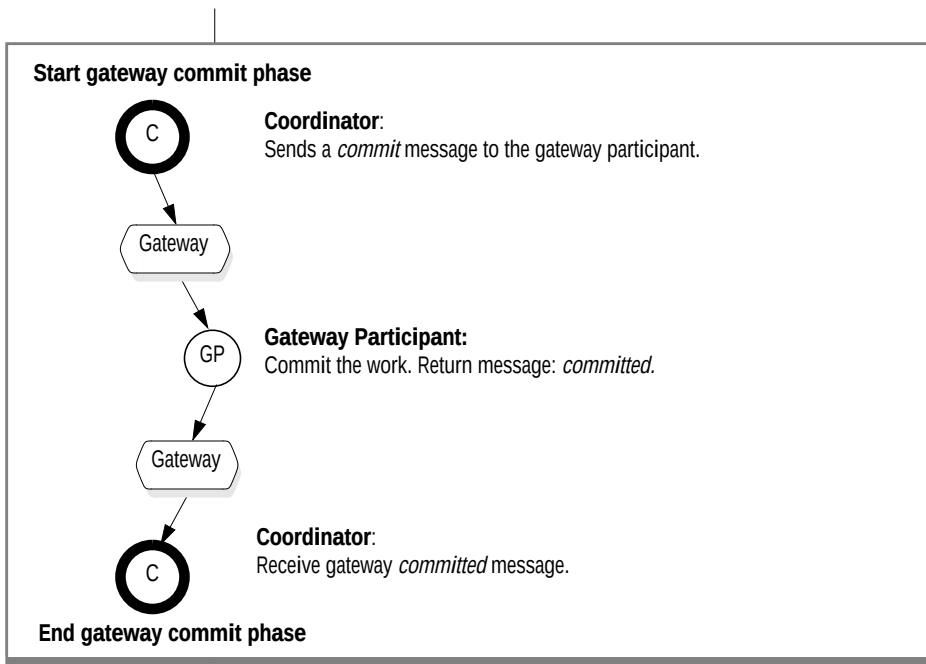


Figure 34-11
*Heterogeneous
Commit Phase That
Results in a
Committed
Transaction*

If the gateway fails to commit the transaction, the coordinator rolls back the entire transaction as Figure 34-11 illustrates.

Heterogeneous Commit Optimization

Universal 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 Universal Server processes using heterogeneous commit, the coordinator or any number of participants can fail. Universal 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 Universal 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, Universal 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 <code>commit transaction</code> message from coordinator, but before participant performs <code>commit</code>	Data consistency is maintained.
After participant receives <code>commit transaction</code> 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 Universal 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 34-44](#).

Interpreting Heterogeneous Commit Error Messages

When Universal 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

Universal 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.
```

Implications of a Failed Heterogeneous Commit

After Universal 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.

Recovering Manually from Failed Two-Phase Commit

Procedure to Determine If Manual Recovery Is Required	35-3
Determine If a Transaction Was Implemented Inconsistently	35-4
Global Transaction Killed Prematurely	35-4
Heuristic End Transaction	35-4
Heuristic Rollback	35-5
Determine If the Distributed Database Contains Inconsistent	
Data	35-6
Obtain Information from the Logical Log	35-7
The Global Transaction Identifier	35-8
Decide If Action Is Needed to Correct the Situation	35-10
Example of Manual Recovery	35-11

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 34-18.](#)) 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 34-20.](#)) You can check for an inconsistent transaction by first examining the INFORMIX-Universal 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 34-26.](#)) 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 Universal 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 did a heuristic rollback:

```
-698 Inconsistent transaction. Number and names of  
servers rolled back.
```

- Examine the messages in the Universal Server message-log file.

If a database inconsistency is possible because of a heuristic decision at a participating Universal Server, the following message appears in the Universal 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 Universal Server 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 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 Universal Server that is assumed to have committed the transaction actually modified data. A read-only Universal 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 Universal 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 Universal Server 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 Universal Server database server. The GTRID is described in [“The Global Transaction Identifier” on page 35-8](#).

3. Obtain a list of the other participants from the coordinator log.
 - a. Examine the log records on the coordinator Universal 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 Universal Server 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 Universal Server 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.

Determine If the Distributed Database Contains Inconsistent Data

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 35-1 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 35-1
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 35-2 with that of the BEGPREP record in Figure 35-1.

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 35-2
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 Universal Server message log and the logical log to locate affected records. Without detailed knowledge of the application, messages are not enough to determine what has happened. Based on your knowledge of your application and your system, you must determine whether to roll back or to commit the transaction. You must also program the compensating transaction that will perform the rollback or the commit.

Example of Manual Recovery

This example illustrates the kind of work that is involved in manual recovery. The following SQL statements were executed by user **nhowe**. Error -698 was returned.

```
%dbaccess
CREATE DATABASE tmp WITH LOG;
CREATE TABLE t (a int);
CLOSE DATABASE;
CREATE DATABASE tmp@apex WITH LOG;
CREATE TABLE t (a int);
CLOSE DATABASE;
DATABASE tmp;
BEGIN WORK;
INSERT INTO t VALUES (2);
INSERT INTO tmp@apex:t VALUES (2);
COMMIT WORK;
### return code -698
```

The following excerpt is taken from the logical log at the current database server:

addr	len	type	xid	id	addr	user
17018	16	CKPOINT	0	0	13018	0
18018	20	BEGIN	2	1	0	08/27/91 10:56:57 3482 nhowe
1802c	32	HINSERT	2	0	18018	1000018 102 4
1804c	40	CKPOINT	0	0	17018	1
begin			xid	id	addr	user
1			2	1	1802c	nhowe
19018	72	BEGPREP	2	0	1802c	6d69 1
19060	16	COMMIT	2	0	19018	08/27/91 11:01:38
1a018	16	ENDTRANS	2	0	19060	580543

The following excerpt is taken from the logical log at the Universal Server database server **apex**:

addr	len	type	xid	id	addr	user
16018	20	BEGIN	2	10	08/27/91 10:57:07 3483	pault
1602c	32	HINSERT	2	0	16018	1000018 102 4
1604c	68	PREPARE	2	0	1602c	eh
17018	16	HEURTX	2	0	1604c	1
17028	12	CLR	2	0	1602c	
17034	16	ROLLBACK	2	0	17018	08/27/91 11:01:22
17044	40	CKPOINT	0	0	15018	1
begin			xid	id	addr	user
1			2	1	17034	-----
18018	16	ENDTRANS	2	0	17034	8806c3

First, you would try to match the transactions in the current database server log with the transactions in the **apex** database server log. The BEGPREP and PREPARE log records each contain the GTRID. You can extract the GTRID by using **onlog -l** and looking at the data portion of the BEGPREP and PREPARE log records. The GTRID is offset 22 bytes into the data portion and is 68 bytes long. A more simple, though less precise, approach is to look at the time of the COMMIT or ROLLBACK records. The times should be close, although there is a slight delay because of the time taken to transmit the commit (or rollback) message from the coordinator to the participant. (This second approach lacks precision because concurrent transactions could commit at the same time although concurrent transactions from one coordinator would probably not commit at the same time.)

To correct this 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
```


Section IX



ON-Monitor

Using ON-Monitor	36-3
Help and Navigation Within ON-Monitor	36-4
Executing Shell Commands from Within ON-Monitor	36-4
ON-Monitor Screen Options	36-4
Setting Configuration Parameters with ON-Monitor	36-12

This section serves as a quick reference for the INFORMIX-Universal Server ON-Monitor screens. You can use it to determine the purpose and use of a specific screen or option.

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 36-1 on page 36-5 through Figure 36-7 on page 36-11).

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: !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 36-1
Status Menu

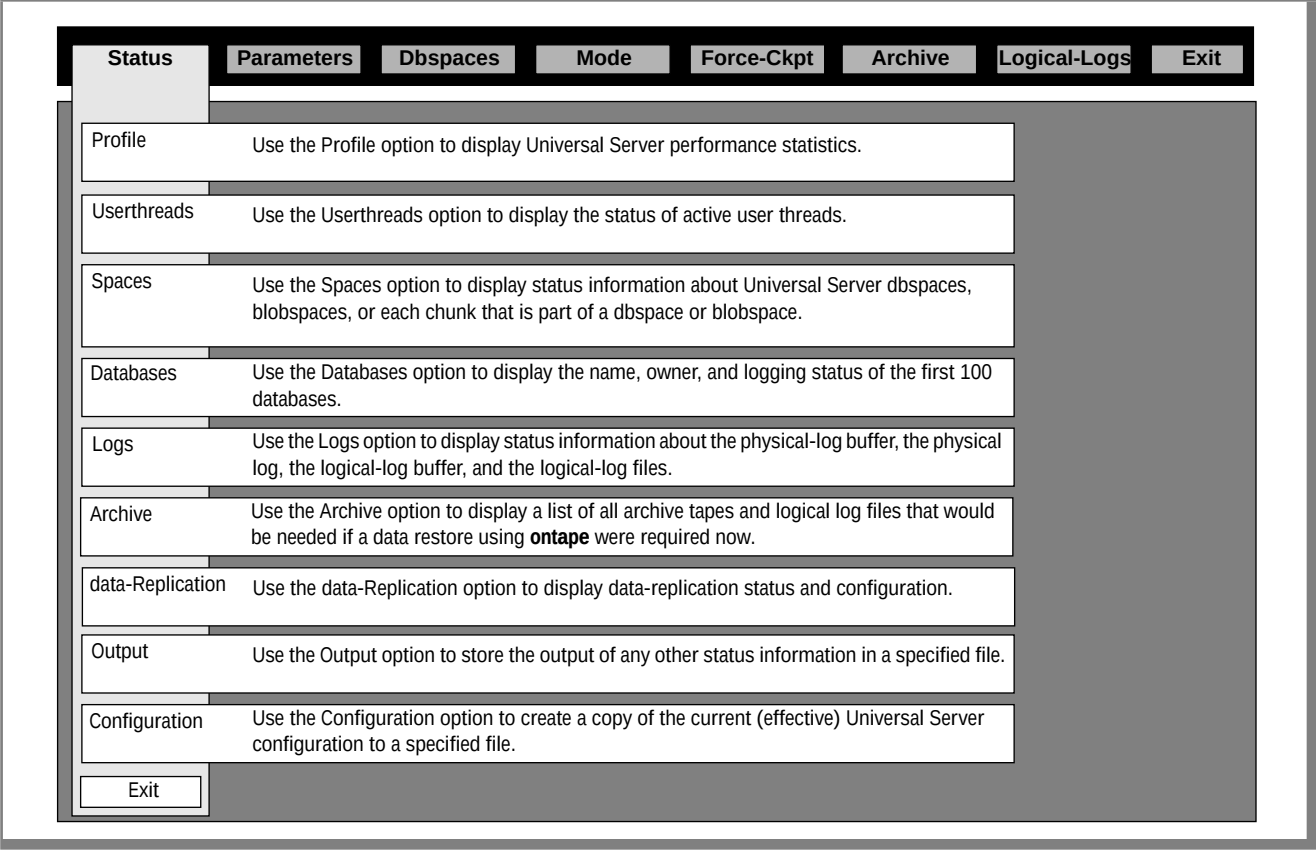


Figure 36-2
Parameters Menu

Status	Parameters	Dbspaces	Mode	Force-Ckpt	Archive	Logical-Logs	Exit
	Initialize	Use the Initialize option to initialize Universal Server disk space or to modify Universal Server disk-space parameters.					
	Shared-Memory	Use the Shared-Memory option to initialize Universal Server shared memory or to modify Universal Server shared-memory parameters.					
	perFormance	Use the perFormance option to specify the number of virtual processors for each VP class.					
	data-Replication	Use the data-Replication option to specify the data-replication parameters.					
	diaGnostics	Use the diaGnostics option to specify values for the diagnostics parameters.					
	pdQ	Use the pdQ option to change parameters for parallel database queries.					
	Add-Log	Use the Add-Log option to add a logical-log file to an Universal Server dbspace.					
	Drop-Log	Use the Drop-Log option to drop a logical log file from an Universal Server dbspace.					
	Physical-Log	Use the Physical-Log option to change the size or the location of the Universal Server physical log.					
	Exit						

Figure 36-3
Dbspaces Menu

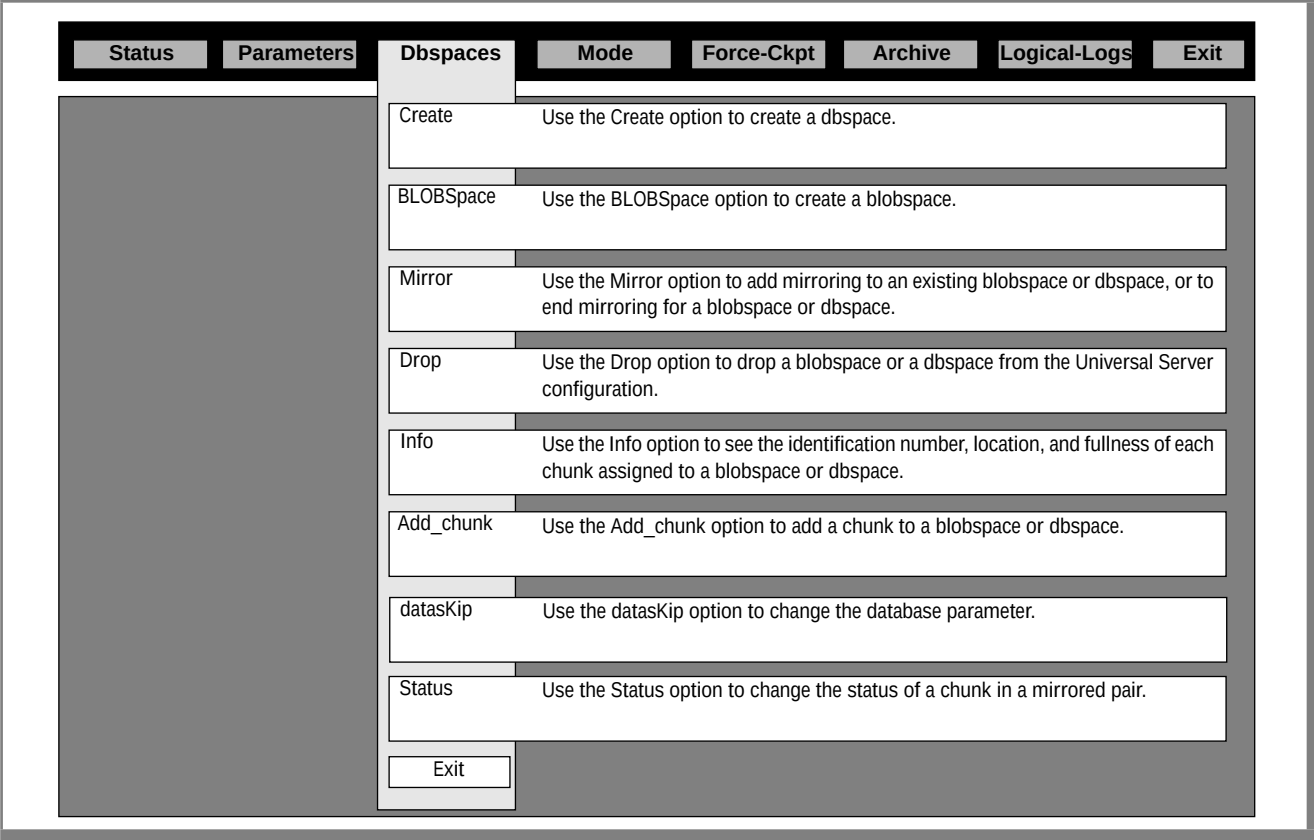


Figure 36-4
Mode Menu

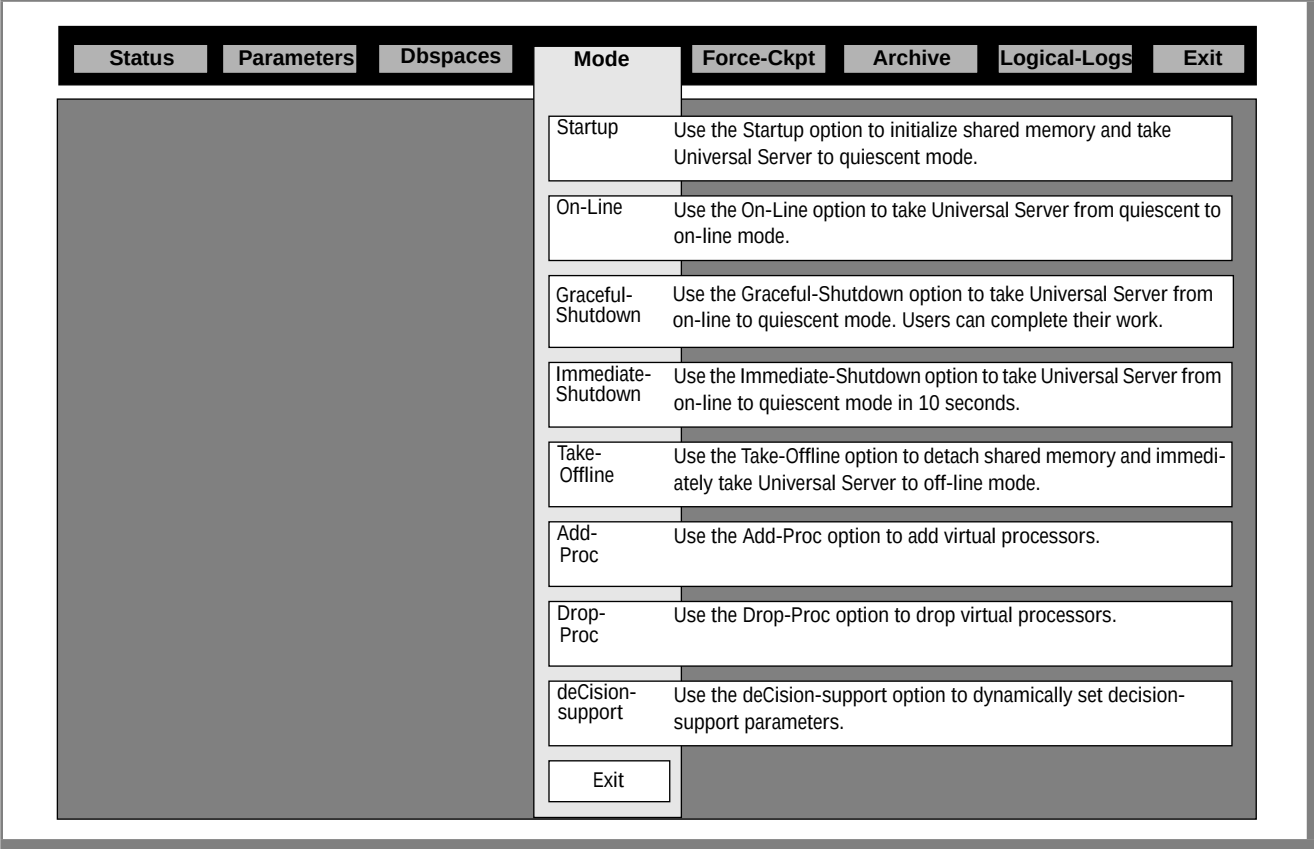


Figure 36-5
Force-Ckpt Option



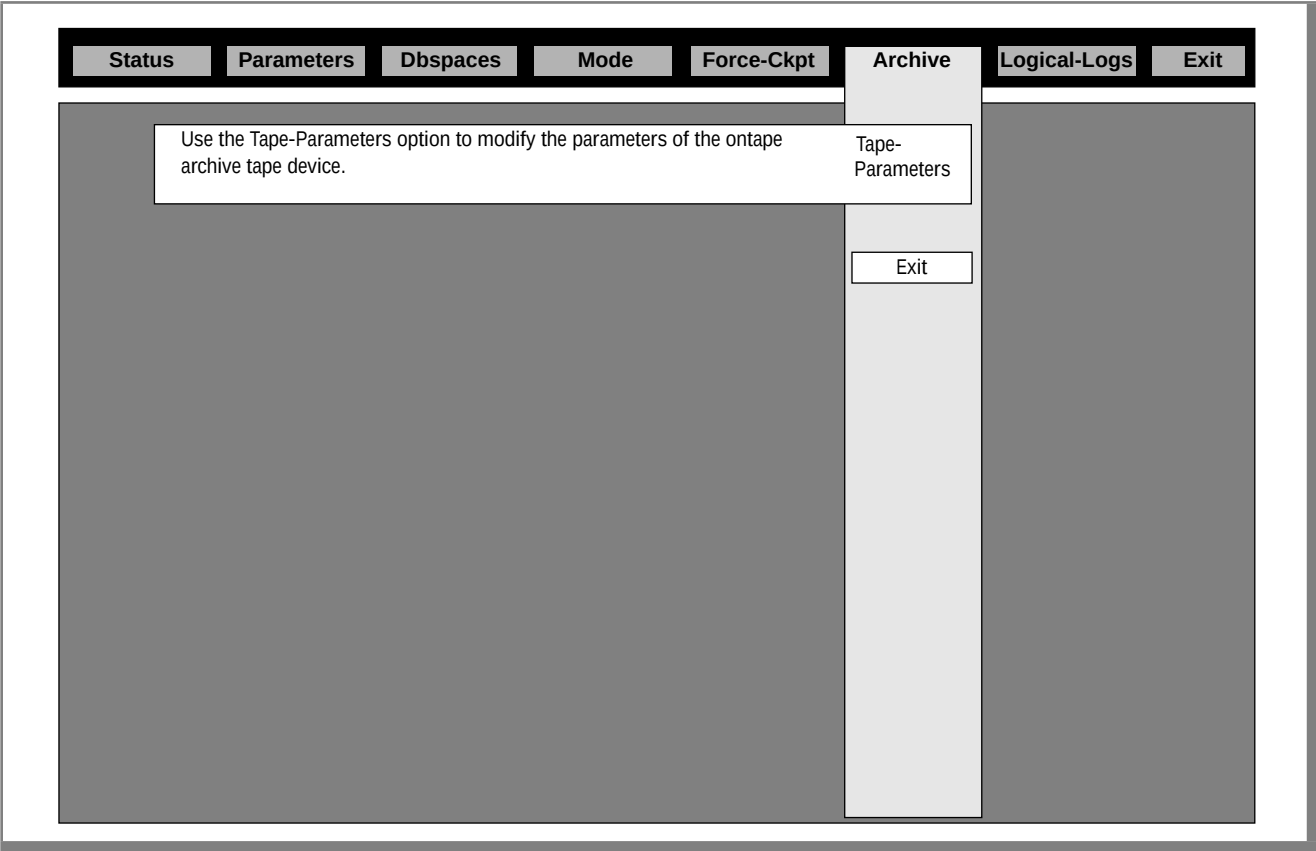
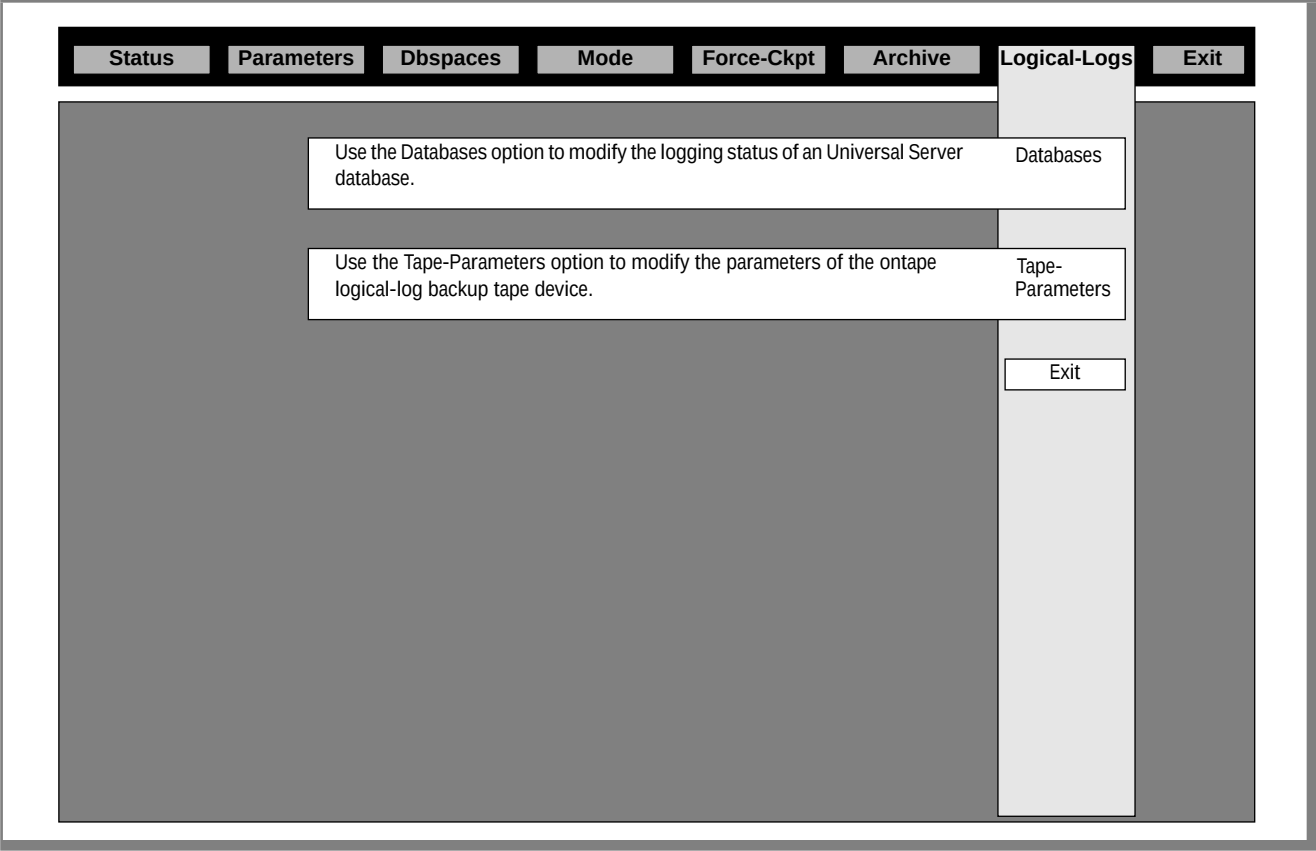


Figure 36-6
Archive Menu

Figure 36-7
Logical-Logs Menu



Setting Configuration Parameters with ON-Monitor

Each ON-Monitor parameters screen is represented here by a pair of examples (Figure 36-8 through Figure 36-18). 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.

Figure 36-8
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      [  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/online_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 36-9
INITIALIZATION
Screen Showing
Parameter Names

```
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 36-10
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           [  0]      Server Name [a_server      ]
Server Aliases [
Dbpace 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      Dbpace 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-OnLine.
    
```

Figure 36-11
SHARED MEMORY
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      {SERVERNUM}      Server Name      {DBSERVERNAME}
Server Aliases     {DBSERVERALIASES}
Dbpace 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}      Dbpace 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 {LTXEHW}
                                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-OnLine.
    
```

Setting Configuration Parameters with ON-Monitor

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 36-12
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 36-13
PERFORMANCE
Screen Showing
Parameter Names



Figure 36-14
*DATA REPLICATION
Screen*

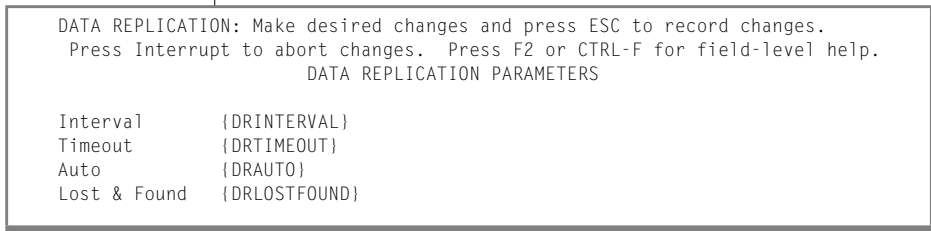


Figure 36-15
*DATA REPLICATION
Screen Showing
Parameter Names*

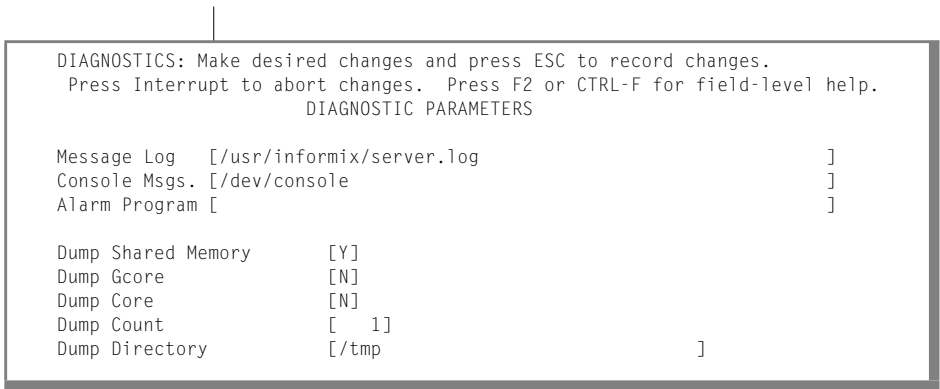


Figure 36-16
DIAGNOSTICS
Screen

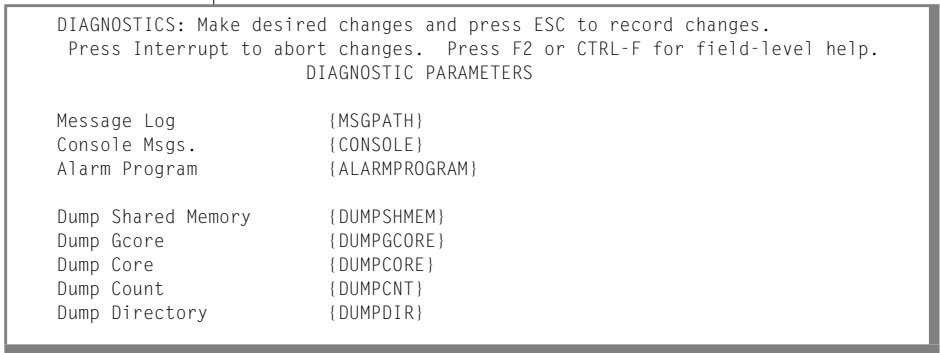


Figure 36-17
DIAGNOSTICS
Screen Showing
Parameter Names

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 36-18
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 36-19
PDQ Screen
Showing Parameter
Names

Configuration Parameters

ONCONFIG Parameters	37-5
ONCONFIG File Conventions	37-6
AFF_NPROCS	37-7
AFF_SPROC	37-8
ALARMPROGRAM	37-9
BUFFERS	37-9
CKPTINTVL	37-11
CLEANERS	37-12
CONSOLE.	37-12
DATASKIP.	37-13
DBSERVERALIASES	37-14
DBSERVERNAME	37-15
DBSPACETEMP.	37-16
DEADLOCK_TIMEOUT.	37-18
DRAUTO	37-18
DRINTERVAL	37-19
DRLOSTFOUND	37-20
DRTIMEOUT.	37-20
DS_MAX_QUERIES	37-21

DS_MAX_SCANS	37-22
DS_TOTAL_MEMORY	37-23
DUMPCNT	37-27
DUMPCORE	37-27
DUMPDIR	37-28
DUMPGCORE	37-28
DUMPSHMEM	37-29
FILLFACTOR	37-30
HETERO_COMMIT	37-30
LBU_PRESERVE	37-31
LOCKS	37-32
LOGBUFF	37-33
LOGFILES	37-34
LOGSIZE	37-35
LOGSMAX.	37-36
LRUS.	37-37
LRU_MAX_DIRTY	37-37
LRU_MIN_DIRTY	37-38
LTAPEBLK.	37-38
LTAPEDEV	37-39
LTAPESIZE	37-40
LTXEHWM	37-41
LTXHWM	37-42
MAX_PDQPRIORITY	37-43
MIRROR	37-44

MIRROROFFSET	.37-45
MIRRORPATH	.37-45
MSGPATH	.37-46
MULTIPROCESSOR	.37-46
NETTYPE.	.37-47
NOAGE	.37-49
NUMAIOVPS	.37-50
NUMCPUVPS	.37-51
OFF_RECVRY_THREADS	.37-52
ON_RECVRY_THREADS	.37-52
ONDBSPACEDOWN.	.37-53
OPCACHEMAX	.37-54
OPTCOMPIND.	.37-54
PHYSBUFF	.37-56
PHYSDBS.	.37-57
PHYSFILE	.37-57
RA_PAGES	.37-58
RA_THRESHOLD.	.37-59
RESIDENT	.37-60
ROOTNAME	.37-60
ROOTOFFSET	.37-61
ROOTPATH	.37-61
ROOTSIZE	.37-62
SBSPACENAME	.37-62
SERVERNUM	.37-63

SHMADD	37-64
SHMBASE	37-65
SHMTOTAL	37-65
SHMVIRTSIZE	37-67
SINGLE_CPU_VP	37-68
STACKSIZE	37-69
STAGEBLOB	37-70
TAPEBLK	37-70
TAPEDEV	37-71
TAPESIZE	37-73
TXTIMEOUT	37-73
USEOSTIME	37-74
VPCLASS	37-75

T

his chapter contains a list of the parameters in the ONCONFIG file and a short discussion of each parameter.

ONCONFIG Parameters

In the discussions in this chapter, one or more of the following attributes (if relevant) is listed for each parameter:

<i>onconfig.std</i> <i>value</i>	The initial value that appears in the onconfig.std file
<i>if not present</i>	The value that INFORMIX-Universal Server supplies if the parameter is missing from your ONCONFIG file
<i>units</i>	The units in which the parameter is expressed
<i>separators</i>	The separator(s) that can be used when the parameter value has several parts. Do <i>not</i> use white space within a parameter value.
<i>range of values</i> <i>takes effect</i>	The legal values for this parameter The time at which a change to the value of the parameter actually affects the operation of Universal Server.
<i>ON-Monitor</i>	The series of menu choices and the item name that displays this parameter in the ON-Monitor utility
<i>utility</i>	The command-line utility that you can use to change the value of the parameter
<i>refer to</i>	Cross reference to further discussion

ONCONFIG File Conventions

The file **\$INFORMIXDIR/etc/\$ONCONFIG** is called the ONCONFIG configuration file or simply the ONCONFIG file. 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 syntax of a parameter line is as follows:

```
PARAMETER_NAME      parameter_value      # comment
```

Parameters and their values in the ONCONFIG file are case sensitive. The parameter names are always all uppercase letters. If the value entry is described with uppercase letters, you must use uppercase (for example, the CPU value of the NETTYPE parameter). You must put white space (tabs or spaces or both) between the parameter name, parameter value, and optional comment. Do not use any tabs or spaces within a parameter value.

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.

For information about the ADTERR, ADTMODE, ADTPATH, and ADTSIZE auditing configuration parameters, refer to the [*INFORMIX-Universal Server Trusted Facility Manual*](#).

AFF_NPROCS

<i>onconfig.std</i>	0
<i>value</i>	
<i>range of values</i>	0 to number of CPUs in the computer
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, perFormance, Num Procs to Affinity
<i>refer to</i>	“CPU Virtual Processors” on page 10-19

On multiprocessor computers that support *processor affinity*, AFF_NPROCS specifies the number of CPUs to which Universal 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. For more information, refer to [“AFF_SPROC” on page 37-8](#).

Informix suggests that you use VPCLASS instead of AFF_NPROCS to specify the number of CPUs. You cannot use both AFF_NPROCS and VPCLASS cpu in the same ONCONFIG file. For information about VPCLASS, refer to [“VPCLASS” on page 37-75](#).

AFF_SPROC

onconfig.std	0
<i>value</i>	
<i>range of values</i>	0 to the value of (AFF_NPROCS - (number of cpu VPs) + 1)
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, perFormance, Proc num to start with
<i>refer to</i>	“CPU Virtual Processors” on page 10-19

On multiprocessor computers that support *processor affinity*, AFF_SPROC specifies the CPU, starting with 0, on which Universal Server starts binding CPU virtual processors to CPUs. The AFF_NPROCS parameter specifies the number of CPUs that Universal 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 will bind CPU virtual processors to CPUs 6, 7, and 8.

Informix suggests that you use VPCLASS instead of AFF_SPROC to specify processor affinity. You cannot use both AFF_SPROC and VPCLASS cpu in the same ONCONFIG file. For information about VPCLASS, refer to [“VPCLASS” on page 37-75](#).

ALARMPROGRAM

<i>onconfig.std</i>	<i>/usr/informix/log_full.sh</i>
<i>value</i>	
<i>range of values</i>	Any valid pathname
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, diaGnostics
<i>refer to</i>	“Event Alarm” on page 33-8

Set ALARMPROGRAM to the full pathname of an executable file that you write and that Universal Server executes when noteworthy events occur. Noteworthy events include database, table, index, or large object failure; chunk or dbspace taken off-line; internal subsystem failure; initialization failure; and detection of long transaction. For a complete list of events that the database server considers noteworthy, refer to [“Event Alarm” on page 33-8](#).

BUFFERS

<i>onconfig.std</i>	<i>200</i>
<i>value</i>	
<i>range of values</i>	Minimum: 100 Maximum: No more than 768 kilobytes (768* 1024)
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Shared_Memory, Max # of Buffers
<i>refer to</i>	“Shared-Memory Buffer Pool” on page 12-24

BUFFERS specifies the maximum number of shared-memory buffers that Universal 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 users’ 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 multiplied by the system page size) to 20 percent of physical memory.

If your databases contain smart large objects, you need to consider them in calculating the value for BUFFERS because smart large objects pass through the regular shared-memory buffer pool. Informix recommends the following formula for calculating the minimum impact of smart large objects on the value of BUFFERS:

```
BUFFERS = BUFFERS + (Server page size in kilobytes *  
concurrently open smart large objects) + desired amount of  
large object user-data to buffer
```

If you also want to do read-ahead, then you should increase the value of BUFFERS further. For more information on read-ahead, see [“RA_PAGES” on page 37-58](#) and [“RA_THRESHOLD” on page 37-59](#). For more information about calculating the value of BUFFERS, refer to Chapter 2 of the *INFORMIX-Universal Server Performance Guide*.

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 obtain the system page size through ON-Monitor under the Parameters:Shared-Memory option, which does not require Universal Server to be running, and also under the Parameters:Initialize option. The command **oncheck -pr**, which checks the root-dbspace reserved pages, also displays the system page size in the first section of its output. The command **onstat -b** also displays the system page size, given as buffer size.

CKPTINTVL

<i>onconfig.std</i>	300
<i>value</i>	
<i>units</i>	seconds
<i>range of values</i>	≥ 0
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, perFormance, Checkpoint Interval
<i>refer to</i>	INFORMIX-Universal Server 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. For a list of the events that trigger a checkpoint, refer to [“Checkpoints” on page 12-56](#).

CLEANERS

<i>onconfig.std</i>	1
<i>value</i>	
<i>range of values</i>	1 to 128
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Shared-Memory, Number of Page Cleaners
<i>refer to</i>	“How the Database Server Flushes Data to Disk” on page 12-45

CLEANERS specifies the number of page-cleaner threads available during Universal 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

<i>onconfig.std</i>	<i>/dev/console</i>
<i>value</i>	
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, diaGnostics, Console Msgs.
<i>refer to</i>	“What Is the Console?” on page 33-12

CONSOLE specifies the pathname destination for console messages.

DATASKIP

onconfig.std	none
value	
if not present	OFF
range of values	ALL, OFF, ON
separators	space
ON-Monitor	Parameters menu, pdQ screen
utility	onspaces -f ; see page 39-74 .
refer to	“Skipping Inaccessible Fragments” on page 16-21

The DATASKIP parameter lets you avoid points of media failure. This capability can result in higher availability for your data. To instruct the database server to skip over 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/COBOL and INFORMIX-ESQL/C.

Use the following syntax in the parameter line:

```
DATASKIP OFF
DATASKIP ON dbspace1 dbspace2...
DATASKIP ALL
```

The Universal Server administrator can alter the value of the configuration parameter at runtime by using the **-f** option of the **onspaces** utility.

An application can use the SQL statement SET DATASKIP to override the DATASKIP value set by the ONCONFIG parameter or by the **onspaces** utility. 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

<i>onconfig.std</i>	none
<i>value</i>	
<i>if not present</i>	none
<i>separators</i>	comma
<i>range of values</i>	18 or fewer lowercase characters
<i>takes effect</i>	When shared memory is initialized. In addition, the sqlhosts file of each database server might need to be updated.
<i>ON-Monitor</i>	Parameters, Shared-Memory, Server Aliases
<i>refer to</i>	DBSERVERNAME and “The DBSERVERALIASES Configuration Parameter” on page 4-45

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. The DBSERVERALIASES configuration parameter lets you assign multiple aliases to a database server, so that each entry in the **sqlhosts** file can have a unique name. For more information about DBSERVERALIASES, refer to [“Using Multiple Connection Types” on page 4-50](#).

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	none
value	
if not present	hostname (name of the host computer)
range of values	18 or fewer lowercase characters. The first character must be a lowercase letter. DBSERVERNAME can include any printable character <i>except</i> the following characters: ■ Uppercase characters ■ A field delimiter (space or tab) ■ A new-line character ■ A comment character
takes effect	When shared memory is initialized. The sqlhosts file 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.
ON-Monitor refer to	Parameters, Shared-Memory, Server Name “The DBSERVERNAME Configuration Parameter” on page 4-44

DBSERVERNAME specifies a unique name associated with this specific occurrence of Universal Server.

The value of DBSERVERNAME is called the *dbservername*. Each *dbservername* is associated with a communication protocol in the **sqlhosts** file. 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.

The default name of the database server is the *hostname* of the computer on which the database server is installed. Informix recommends that you assign a name using DBSERVERNAME instead of using the default name because of possible conflict with other database servers on the same host computer

DBSPACETEMP

<i>onconfig.std</i>	none
<i>value</i>	
<i>if not present</i>	The dbspace name specified by ROOTNAME
<i>range of values</i>	Length of list cannot exceed 254 characters
<i>separators</i>	Comma or colon (no white space)
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Shared-Memory, Dbspace Temp
<i>refer to</i>	“Where Are Temporary Tables Stored?” on page 14-29

The DBSPACETEMP configuration parameter 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. For more information, refer to [“What Is a Temporary Table?” on page 14-28](#).

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.

The **PSORT_DBTEMP** environment variable can be used to specify a list of UNIX 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 the [INFORMIX-Universal Server Performance Guide](#).

If an individual client application wishes 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. For more information on the **PSORT_DBTEMP** and **DBSPACETEMP** environment variables, refer to Chapter 4 of the *Informix Guide to SQL: Reference*.

When Changes to DBSPACETEMP Take Effect

When you create a temporary dbspace with ON-Monitor or **onspaces**, Universal Server does not use the newly created temporary dbspace until you take both of 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

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.

DEADLOCK_TIMEOUT

<i>onconfig.std</i>	60
<i>value</i>	
<i>units</i>	Seconds
<i>range of values</i>	Positive integers
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Shared-Memory, Deadlock Timeout
<i>refer to</i>	“Configuration Parameters Used in Two-Phase Commits” on page 34-37

DEADLOCK_TIMEOUT specifies the maximum number of seconds that a 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

<i>onconfig.std</i>	0
<i>value</i>	
<i>range of values</i>	0 OR OFF; do not automatically switch 1 OR RETAIN_TYPE; automatically switch secondary to standard on a data-replication failure. Switch back to secondary when restarting data replication. 2 OR REVERSE_TYPE; automatically switch secondary to standard on a data-replication failure. Switch to primary (and switch original primary to secondary) when restarting data replication.
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, data-Replication, Auto
<i>refer to</i>	“What Is Automatic Switchover?” on page 29-18

DRAUTO determines how a secondary database server reacts to a data-replication failure. This parameter should have the same value on both data-replication servers.

If DRAUTO is set to OFF, the secondary database server remains a secondary database server in read-only mode when a 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 data-replication failure is detected. If DRAUTO is set to RETAIN_TYPE, the original secondary database server switches back to type secondary when the data-replication connection is restored. If DRAUTO is set to REVERSE_TYPE, the original secondary database server switches to type *primary* when the data-replication connection is restored, and the original primary switches to type secondary.

Use this parameter with care, though, because a network failure (that is, when the primary database server does not really fail, but the secondary database server perceives network slowness as a data-replication failure) can cause the two database servers to become out of synch.

DRINTERVAL

<i>onconfig.std</i>	30
<i>value</i>	
<i>units</i>	Seconds
<i>range of values</i>	-1, 0, and positive integer values
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, data-Replication, Interval
<i>refer to</i>	“When Are Log Records Sent?” on page 29-10

DRINTERVAL specifies the maximum time interval in seconds between flushing of the data-replication buffer.

To update synchronously, set the parameter to -1. For a discussion of the differences between asynchronous and synchronous updating, refer to [“When Are Log Records Sent?” on page 29-10](#).

DRLOSTFOUND

<i>onconfig.std</i>	/usr/etc/dr.lostfound
<i>value</i>	
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, data-Replication, Lost & Found
<i>refer to</i>	“Lost-and-Found Transactions” on page 29-11

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

<i>onconfig.std</i>	30
<i>value</i>	
<i>units</i>	seconds
<i>range of values</i>	Positive integers
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, data-Replication, Timeout
<i>refer to</i>	“How Are Data-Replication Failures Detected?” on page 29-16

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. You can calculate DRTIMEOUT for your data replication pair as follows:

$$\text{DRTIMEOUT} = \text{wait\_time} / 4$$

For example, suppose you determine that the database servers in your data replication pair must wait 160 seconds before they assume that a data replication failure occurred. Use the preceding formula to set DRTIMEOUT as follows:

$$\text{DRTIMEOUT} = 160 \text{ seconds} / 4 = 40 \text{ seconds}$$

DS_MAX_QUERIES

onconfig.std	none
value	
if not present	(number of cpu vps)*2* 128
range of values	Minimum = 1; maximum = 8 * 1024 * 1024
ON-Monitor	Parameters menu, pdQ screen
utility	onmode -Q ; see page 39-46 .
refer to	“Using the Memory Grant Manager” on page 19-12

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:

$$\text{memory\_reserved} = (\text{DS\_TOTAL\_MEMORY} / \text{DS\_MAX\_QUERIES}) * \text{DS\_MAX\_QUERIES} * (\text{PDQPRIORITY} / 100) * (\text{MAX\_PDQPRIORITY} / 100)$$

For more information on the effects of DS_MAX_QUERIES, refer to Chapter 2, “Configuration Impacts on Performance,” in the [INFORMIX-Universal Server Performance Guide](#).

You can use the **onmode** utility to change the value of DS_MAX_QUERIES while the database server is on-line. For more information about using **onmode**, refer to [“Change Decision-Support Parameters” on page 39-46](#).

DS_MAX_SCANS

onconfig.std value	1,048,576 (that is, 1024 * 1024)
range of values	Between 10 and 1024 * 1024
ON-Monitor	Parameters menu, pdQ screen
utility	onmode -S ; see page 39-46 .
refer to	“Limiting the Number of Concurrent Scans” on page 19-8

DS_MAX_SCANS limits the number of PDQ scan threads that the database server can execute concurrently.

When a user issues a query, Universal Server apportions some number of scan threads, depending on the following:

- 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:

$$\text{reserved\_threads} = \min \left(\text{nfrags}, \left(\text{DS\_MAX\_SCANS} * \frac{\text{PDQPRIORITY}}{100} * \frac{\text{MAX\_PDQPRIORITY}}{100} \right) \right)$$

If the DS_MAX_SCANS formula is greater than or equal to the number of fragments, then the query is held in the ready queue until enough 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 24 (*nfrags*; at 27, the DS_MAX_SCANS formula is larger) 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.

For more information on the effects of DS_MAX_SCANS, see the [INFORMIX-Universal Server Performance Guide](#).

You can use the **onmode** utility to change the values of DS_MAX_SCANS while the database server is on-line. For more information about using **onmode**, refer to [“Change Decision-Support Parameters” on page 39-46](#).

DS_TOTAL_MEMORY

<i>default value</i>	none
<i>if not present</i>	If SHMTOTAL=0 and DS_MAX_QUERIES is set, then DS_MAX_QUERIES * 128. If SHMTOTAL=0 and DS_MAX_QUERIES is not set, then (number of cpu VPs)* 2 * 128. If SHMTOTAL>0, then SHMTOTAL minus 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).
<i>units</i>	kilobytes
<i>range of values</i>	If DS_MAX_QUERY is set, minimum is DS_MAX_QUERY * 128. If DS_MAX_QUERY is not set, then minimum is (number of cpu VPs) * 2 * 128. Maximum is <i>maxmem</i>.
<i>ON-Monitor utility</i>	Parameters menu, pdQ screen
<i>refer to</i>	onmode -M; see page 39-46. “Adjusting the Amount of Memory” on page 19-7

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 Informix buffer-pool size.

Do not confuse DS_TOTAL_MEMORY with configuration parameter SHMTOTAL. For OLTP applications, set DS_TOTAL_MEMORY to between 20 and 50 percent of the value of SHM_TOTAL in kilobytes. For applications that involve large decision-support queries, increase the value of DS_TOTAL_MEMORY to between 50 and 80 percent of SHM_TOTAL. If your Universal Server instance is used exclusively for decision-support queries, set this parameter to 90 percent of SHM_TOTAL.

You can use the **onmode** utility to change the values of DS_TOTAL_MEMORY while the database server is on-line. For more information about using **onmode**, refer to [“Change Decision-Support Parameters” on page 39-46](#).

Algorithm for DS_TOTAL_MEMORY

Universal 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 VPCLASS cpu or 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

Universal 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
```

The variable **min_ds_total_memory** is described in [“Derive a Minimum for Decision-Support Memory” on page 37-24](#).

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.

DUMPCNT

<i>onconfig.std</i>	1
value	
if not present	1
range of values	Positive integers
takes effect	When shared memory is initialized
ON-Monitor	Parameters, diaGnostics, Dump Count
refer to	“Collecting Diagnostic Information” on page 31-10

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. 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");
```

DUMPCORE

<i>onconfig.std</i>	0
value	
range of values	0 = do not dump core; 1 = dump core
takes effect	When shared memory is initialized
ON-Monitor	Parameters, diaGnostics, Dump Core
refer to	“Collecting Diagnostic Information” on page 31-10

DUMPCORE controls whether assertion failures cause a virtual processor to dump core. The core file is left in the directory from which Universal Server was last invoked. (The DUMPPDIR 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

<i>onconfig.std</i>	/tmp
<i>value</i>	
<i>if not present</i>	/tmp
<i>range of values</i>	Any directory to which user informix has write access
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, diaGnostics, Dump Directory
<i>refer to</i>	“Collecting Diagnostic Information” on page 31-10

DUMPDIR specifies a directory where dumps of shared memory, gcore files, or messages from a failed assertion are placed.

Because shared memory can be large, set DUMPDIR to a file system with a significant amount of space.

DUMPGCORE

<i>onconfig.std</i>	0
<i>value</i>	
<i>range of values</i>	0 = do not dump gcore; 1 = dump gcore
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, diaGnostics, Dump Gcore
<i>refer to</i>	“Collecting Diagnostic Information” on page 31-10

DUMPGCORE is used with operating systems that support the **gcore** utility. 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

<i>onconfig.std</i>	1
<i>value</i>	
<i>range of values</i>	0 = do not dump shared memory 1 = dump shared memory
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, diaGnostics, Dump Shared Memory
<i>refer to</i>	“Collecting Diagnostic Information” on page 31-10

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 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

<i>onconfig.std</i>	90
<i>value</i>	
<i>units</i>	Percent
<i>range of values</i>	1 through 100
<i>takes effect</i>	When the index is built. Existing indexes are not changed. To use the new value, the indexes must be rebuilt.
<i>ON-Monitor</i>	Parameters, Shared-Memory, Index Page Fill Factor
<i>refer to</i>	“Structure of B-Tree Index Pages” on page 42-48

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

<i>onconfig.std</i>	<i>value</i>
<i>default value</i>	0
<i>range of values</i>	1 = Heterogeneous commit enabled 0 = Heterogeneous commit disabled
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Shared-Memory, Heterogeneous Commit
<i>refer to</i>	“Heterogeneous Commit Protocol” on page 34-38

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.

For more information about the Informix gateway, refer to the *INFORMIX-Enterprise Gateway Manager User Manual*.

LBU_PRESERVE

<i>onconfig.std</i>	0
value	
range of values	1 to enable the logs-full high-water mark; 0 to disable the logs-full high-water mark
takes effect	When shared memory is initialized.
ON-Monitor	Parameters, Shared-Memory, Preserve Log for Log Backup
refer to	“Enabling the Logs-Full High-Water Mark” on page 22-10

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	2000
value	
range of values	Minimum: 2000 Maximum: 8,000,000
takes effect	When shared memory is initialized
ON-Monitor	Parameters, Shared-Memory, Max # of Locks
refer to	“Lock Table” on page 12-21; “How Database Server Threads Access Shared Buffers” on page 12-36

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.”

LOGBUFF

onconfig.std	32
value	
units	kilobytes
range of values	$2 * \text{page size} \leq n \leq \text{LOGSIZE}$
takes effect	When shared memory is initialized
ON-Monitor	Parameters, Shared-Memory, Logical Log Buffer Size
refer to	“Logical-Log Buffer” on page 12-26

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. If you log smart-large-object columns, you must ensure that the log buffer is big enough to accommodate your largest large-object page size plus one server page. Universal Server only logs the portion of a smart-large-object page that changed.

The system page size is platform-dependent. You can obtain the system page size through ON-Monitor under the Parameters:Shared-Memory option, which does not require the database server to be running, and also under the Parameters:Initialize option. The command **oncheck -pr**, which checks the root-dbspace reserved pages, also displays the system page size in the first section of its output.



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

<i>onconfig.std</i>	6
<i>value</i>	
<i>if not present</i>	6
<i>range of values</i>	Integer values from 3 to LOGSMAX (inclusive)
<i>takes effect</i>	The value is read and used during disk initialization but updated when you add a new log file using ON-Monitor or the onparams utility.
<i>ON-Monitor</i>	Parameters, Initialize, Number of Logical Logs
<i>utility</i>	onparams ; see page 39-50 .
<i>refer to</i>	“What Should Be the Size and Number of Logical-Log Files?” on page 22-12

LOGFILES specifies the number of logical-log files created by Universal Server during disk initialization.

If you want to change the number of logical-log files, add or drop files using the procedures in [“Adding a Logical-Log File” on page 23-4](#) and [“Dropping a Logical-Log File” on page 23-6](#). If you add or drop log files using ON-Monitor or the **onparams** utility, LOGFILES is updated automatically.

LOGSIZE

onconfig.std	1500
value	
if not present	1500
units	kilobytes
range of values	Minimum = 200; maximum = (ROOTSIZE - PHYSFILE - 512 - (63 * ((pagesize) / 1024))) / LOGFILES; pagesize = platform-specific 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.
ON-Monitor	Parameters, Initialize, Log. Log Size
refer to	“What Should Be the Size and Number of Logical-Log Files?” on page 22-12

LOGSIZE specifies the size that is used when logical-log files are created. It does not change the size of existing logical-log files.

To change the size of logical-log files, follow the procedure in [“Changing LOGSIZE or LOGFILES” on page 23-10](#).

The **onconfig.std** value of LOGSIZE (1500 kilobytes) and the **onconfig.std** value of LOGFILES (6) mean that the total logical-log size that Universal Server creates with those values is 9,000 kilobytes. This value might not be appropriate for your system. For a general discussion of logical-log size considerations, refer to [“How Big Should the Logical Log Be?” on page 22-5](#).



Important: *If you declare logging for a smart-large-object column, you must ensure that the logical log is considerably larger than the amount of data logged during inserts or updates. The logical-log size must exceed the total amount of data logged while a transaction is active, even if the user-data portion of the smart large object is not logged. Otherwise, you increase your chances of incurring a long-transaction condition because multimedia updates can collect data over a long period of time.*

To verify the *pagesize* that the database server uses on your platform, see the last line of output from the **onstat -b** command.

LOGSMAX

<i>onconfig.std</i>	6
<i>value</i>	
<i>if not present</i>	3
<i>range of values</i>	Integers from the value of LOGFILES to 32,767 (inclusive)
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Shared-Memory, Max # of Logical Logs
<i>refer to</i>	“What Should Be the Size and Number of Logical-Log Files?” on page 22-12

LOGSMAX specifies the maximum number of logical-log files that an instance of Universal 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. For instructions, refer to [“Moving a Logical-Log File to Another Dbspace” on page 23-8](#).

To change LOGSMAX, follow the procedure in [“Changing Logical-Log Configuration Parameters” on page 23-10](#).

LRUS

<i>onconfig.std</i>	8
<i>value</i>	
<i>if not present</i>	If MULTIPROCESSOR is set, MAX(4,number of cpu VPs) If MULTIPROCESSOR is not set, 4
<i>range of values</i>	1 to 128 (1 to 512 for 64-bit platforms)
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, perFormance, Num of LRU queues
<i>refer to</i>	“LRU Queues” on page 12-36

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. For more information about tuning the LRU queue, refer to the [INFORMIX-Universal Server Performance Guide](#).

LRU_MAX_DIRTY

<i>onconfig.std</i>	60
<i>value</i>	
<i>units</i>	Percent
<i>range of values</i>	0 to 100
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, perFormance, LRU Max Dirty
<i>refer to</i>	“Limiting the Number of Pages Added to the MLRU Queues” on page 12-39

LRU_MAX_DIRTY specifies the percentage of modified pages in the LRU queues that, when reached, flags the queue to be cleaned. The interaction between page-cleaner threads and the LRU queues is described in [“LRU Queues” on page 12-36](#).

LRU_MIN_DIRTY

onconfig.std	50
value	
units	Percent
range of values	0 to 100
takes effect	When shared memory is initialized
ON-Monitor	Parameters, perFormance, LRU Min Dirty
refer to	“When MLRU Cleaning Ends” on page 12-40

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. The interaction between the page-cleaner threads and the LRU queues is described in [“LRU Queues” on page 12-36](#).

LTAPEBLK

onconfig.std	16
value	
units	kilobytes
range of values	Values greater than $(page\ size)/1024$
takes effect	For ontape , the value is read when the utility is executed. For onload and onunload , new values take effect when shared memory is initialized.
ON-Monitor	Parameters, Initialize, Block Size (follows Log Tape Dev.)
refer to	INFORMIX-Universal Server Archive and Backup Guide for information on how this parameter affects ontape . For information on how this parameter affects onload and onunload , refer to the Informix Migration Guide .

The LTAPEBLK parameter 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**.

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. (The UNIX **dd** utility can verify this fact. It is available with most UNIX systems.) If not, you might not be able to read from the tape.

If you are using **onload** or **onunload**, you can specify a different block size at the command line. For information about **onload** and **onunload**, refer to the [Informix Migration Guide](#).

The system page size is platform dependent. You can obtain the system page size through ON-Monitor under the Parameters:Shared-Memory option, which does not require the database server to be running, and also under the Parameters:Initialize option. The command **oncheck -pr**, which checks the root-dbspace reserved pages, also displays the system page size in the first section of its output.

LTAPEDEV

<i>onconfig.std</i>	/dev/tapedev
<i>value</i>	
<i>if not present</i>	/dev/null
<i>takes effect</i>	The value is read from the ONCONFIG file by ontape when it is executed. For onload and onunload , new values take effect when shared memory is initialized.
<i>ON-Monitor</i>	Parameters, Initialize, Log Tape Dev
<i>refer to</i>	INFORMIX-Universal Server Archive and Backup Guide for information on how this parameter affects ontape . For information on how this parameter affects onload or onunload , refer to the Informix Migration Guide .

The LTAPEDEV parameter 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 [INFORMIX-Universal Server 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. For more information, refer to [“TAPEDEV” on page 37-71](#).

LTAPESIZE

onconfig.std	10240
value	
units	kilobytes
range of values	Positive integers
takes effect	The value is read from the ONCONFIG file by ontape when it is executed. For onload and onunload , new values take effect when shared memory is initialized.
ON-Monitor	Parameters, Initialize, Total Tape Size (follows Log Tape Dev.)
refer to	INFORMIX-Universal Server Archive and Backup Guide for information on how this parameter affects ontape . For information on how this parameter affects onload or onunload , refer to the Informix Migration Guide .

The LTAPESIZE parameter 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. For information about **onload** and **onunload**, refer to the [Informix Migration Guide](#).

LTXEHWMM

<i>onconfig.std</i>	60
<i>value</i>	
<i>units</i>	Percent
<i>range of values</i>	Between the value of LTXHWM and 100
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Shared-Memory, Long TX HWM Exclusive
<i>refer to</i>	“Avoiding Long Transactions” on page 22-19

LTXEHWMM specifies the *long-transaction, exclusive-access, high-water mark*. If the logical log fills to LTXEHWMM, the long transaction currently being rolled back is given *exclusive* access to the logical log. For example, if you set LTXEHWMM 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. For more information about long transactions in the logical log, refer to [“LTXHWM” on page 37-42](#).

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

<i>onconfig.std</i>	50
<i>value</i>	
<i>units</i>	Percent
<i>range of values</i>	From 1 to 100
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Shared-Memory, Long TX HWM
<i>refer to</i>	“Avoiding Long Transactions” on page 22-19

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 LTXEHW 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

<i>onconfig.std</i>	100
<i>value</i>	
<i>if not present</i>	100
<i>range of values</i>	0 to 100
<i>takes effect</i>	On all user sessions
<i>ON-Monitor</i>	Parameters menu, pdQ screen
<i>utility</i>	onmode -D ; see page 39-46 .
<i>refer to</i>	“Controlling the Use of Resources” on page 19-3

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.

Viewed from the perspective of decision support and OLTP, MAX_PDQPRIORITY allows the database administrator to run decision support concurrently with OLTP without a deterioration of OLTP performance. However, if MAX_PDQPRIORITY is set too low, the performance of decision support queries can degrade.

The MAX_PDQPRIORITY configuration parameter takes one of the following values and affects all user sessions. If MAX_PDQPRIORITY is not set, the **onconfig.std** value of MAX_PDQPRIORITY is 100.

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.
<i>number</i>	The <i>number</i> 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.

You can use the **onmode** utility to change the value of MAX_PDQPRIORITY while the database server is on-line. For more information on **onmode**, refer to [“Change Decision-Support Parameters” on page 39-46](#).

MIRROR

<i>onconfig.std</i>	0
<i>value</i>	
<i>range of values</i>	0=do not use mirroring in any dbspace; 1=allow mirroring to be turned on
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Initialize, Mirror
<i>refer to</i>	“Enabling Mirroring” on page 28-4

The MIRROR parameter 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. If you later decide to add mirroring, you can change the parameter value through ON-Monitor or by editing your configuration file.

MIRROROFFSET

<i>onconfig.std</i>	0
<i>value</i>	
<i>units</i>	kilobytes
<i>range of values</i>	$n \geq 0$
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Initialize, Offset (below Mirror Path)
<i>refer to</i>	“Mirroring the Root Dbspace During Initialization” on page 28-7

The MIRROROFFSET parameter 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

<i>onconfig.std</i>	The <i>onconfig.std</i> value is blank.
<i>value</i>	
<i>range of values</i>	65 or fewer characters
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Initialize, Mirror Path
<i>refer to</i>	“Mirroring the Root Dbspace During Initialization” on page 28-7

The MIRRORPATH parameter 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. No ***onconfig.std*** value is provided, but ***/dev/mirror_root*** is one suggestion for a link pathname.

MSGPATH

<i>onconfig.std</i>	<i>/usr/informix/online.log</i>
<i>value</i>	
<i>if not present</i>	<i>/dev/tty</i>
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, diaGnostics, Message Log
<i>refer to</i>	“What Is the Message Log?” on page 33-6

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

<i>onconfig.std</i>	<i>0</i>
<i>value</i>	
<i>if not present</i>	Platform dependent.
<i>range of values</i>	0=no multiprocessor, 1=multiprocessor available
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, perFormance, Multiprocessor Machine
<i>refer to</i>	“CPU Virtual Processors” on page 10-19

MULTIPROCESSOR specifies whether the database server performs locking in a manner that is suitable for a single-processor computer or a multiprocessor computer.

NETTYPE

<i>syntax</i>	Protocol, number of poll threads, number of connections, VP class. The protocol entry is required.
	Unlike most configuration parameters, NETTYPE has four fields, <i>protocol</i> , <i>poll_threads</i> , <i>connections</i> , and <i>VP_class</i> , that you delimit with commas. You cannot use any white space in the fields, but you can omit trailing commas.
<i>onconfig.std values</i>	none
<i>if not present</i>	<i>protocol</i>: nettype field from the sqlhosts file (optionally minus the database server prefix of on or ol) <i>poll_threads</i>: 1 <i>connections</i>: 50 <i>VP_class</i>: NET if it is for DBSERVERALIASES; CPU if it is for DBSERVERNAME
<i>range of values</i>	<i>protocol</i>: same as the nettype values (with or without the database server prefix of on or ol) that are accepted in the sqlhosts file <i>poll_threads</i>: $n \geq 1$ if <i>VP_class</i> is NET; $1 \leq n \leq (\text{number of CPU VPs})$ if <i>VP_class</i> is CPU <i>connections</i>: $1 \leq \text{connections} \leq 32767$ <i>VP_class</i>: CPU or NET
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters: perFormance, Protocol, Threads, Users (<i>connections</i>), VP-class entries for the protocols supported on the computer
<i>refer to</i>	“Network Virtual Processors” on page 10-31 and “The nettype Field” on page 4-27

The NETTYPE parameter provides tuning options for the protocols defined by **dbservername** entries in the **sqlhosts** file. Each **dbservername** entry in the **sqlhosts** file 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 Universal 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).

The protocol entry is the same as the **nettype** field in the **sqlhosts** file, 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. For more information on the possible values for the protocol entry, refer to [“The nettype Field” on page 4-27](#).

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. For more information on poll threads, refer to [“Network Virtual Processors” on page 10-31](#).

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.

You can set the `VP class` entry to specify either `CPU` or `NET`. For advice on whether to run the poll threads on CPU or NET virtual processors, refer to [“Should Poll Threads Run on CPU or Network Virtual Processors?” on page 10-32](#). 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.

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`.

NOAGE

<i>onconfig.std</i>	0
<i>value</i>	
<i>range of values</i>	0=use priority aging; 1=disable priority aging
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, perFormance, Disable Priority Aging
<i>refer to</i>	“Preventing Priority Aging” on page 10-21

Some UNIX operating systems lower the priority of processes as the processes run over a long period of time. NOAGE, when set to one, disables *priority aging* of CPU virtual processors by the operating system. When NOAGE is set to the default, zero, 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.

Informix suggests that you specify priority aging with the VPCLASS parameter instead of the NOAGE parameter. You cannot use both NOAGE and VPCLASS CPU in the same ONCONFIG file. For more information, refer to [“VPCLASS” on page 37-75](#).

NUMAIOVPS

<i>onconfig.std</i>	none
<i>value</i>	
<i>if not present</i>	(2 * <i>number_of_chunks</i>) or 6, whichever is greater; <i>number_of_chunks</i> is the number of chunks you have allocated.
<i>range of values</i>	integer greater than or equal to 1
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, perFormance, AIO VPs
<i>utility</i>	onmode -p ; see page 39-44
<i>refer to</i>	“Asynchronous I/O” on page 10-28

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 Universal Server disk I/O, other than I/O to the log files. If kernel-asynchronous I/O is implemented, the database server uses AIO virtual processors to perform I/O to cooked file spaces.

Informix suggests that you specify the number of AIO VPs with VPCLASS AIO instead of NUMAIOVPS. You cannot use both NUMAIOVPS and VPCLASS AIO in the same ONCONFIG file. For information about VPCLASS, refer to [“VPCLASS” on page 37-75](#).

NUMCPUVPS

<i>onconfig.std</i>	1
<i>value</i>	
<i>range of values</i>	1 to the number of CPUs
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, perFormance, CPU VPs
<i>utility</i>	onmode -p ; see page 39-44
<i>refer to</i>	“CPU Virtual Processors” on page 10-19

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.

Informix suggests that you specify the number of CPU virtual processors with VPCLASS CPU instead of NUMCPUVPS. You cannot use both NUMCPUVPS and VPCLASS CPU in the same ONCONFIG file. For information about VPCLASS, refer to [“VPCLASS” on page 37-75](#).

OFF_RECVRY_THREADS

<i>onconfig.std</i>	10
<i>value</i>	
<i>range of values</i>	Positive integers
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, perFormance, Off-Line Recovery Threads
<i>refer to</i>	INFORMIX-Universal Server Backup and Restore Guide for ON-Bar and to the INFORMIX-Universal Server 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

<i>onconfig.std</i>	1
<i>value</i>	
<i>range of values</i>	Positive integers
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, perFormance, On-Line Recovery Threads
<i>refer to</i>	INFORMIX-Universal Server Backup and Restore Guide for ON-Bar. Refer to the INFORMIX-Universal Server 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

<i>onconfig.std</i>	0
<i>value</i>	
<i>range of values</i>	0, 1, 2
<i>ON-Monitor</i>	Parameters, Shared-Memory, Dbspace Down Option
<i>refer to</i>	“Monitoring the Database Server for Disabling I/O Errors” on page 33-58

ONDBSPACEDOWN defines the action the database server will take when any disabling event occurs on a noncritical dbspace. The following values are valid for this parameter:

- 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.

OPCACHEMAX

onconfig.std	0
value	
if not present	128
units	kilobytes
range of values	0 to (4 * 1024 * 1024)
takes effect	When OnLine / Optical needs more memory
ON-Monitor	Parameters, Initialize, StageBlob
refer to	INFORMIX-OnLine/Optical User Manual

OPCACHEMAX specifies the size of the memory cache for the INFORMIX-OnLine / Optical subsystem. Universal Server stores pieces of large objects in the memory cache before it delivers them to the subsystem. Use this parameter only if you use optical storage with INFORMIX-OnLine / Optical.

OPTCOMPIND

onconfig.std	2
value	
range of values	0, 1, 2
ON-Monitor	Parameters menu, pdQ screen
refer to	“End-User Control of Resources” on page 19-11

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).”



The following values are legal for this variable:

- 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.

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.

For more information on the methods that Universal Server uses to perform a join, refer to Chapter 4 of the [INFORMIX-Universal Server Performance Guide](#).

A client application can also influence the optimizer in its choice of a join method. For more information, refer to the **OPTCOMPIND** environment variable in the [Informix Guide to SQL: Reference](#).

PHYSBUFF

<i>onconfig.std</i>	32
<i>value</i>	
<i>units</i>	kilobytes
<i>range of values</i>	Page size ≤ n ≤ PHYSFILE
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Shared-Memory, Physical-log Buffer Size
<i>refer to</i>	“Physical-Log Buffer” on page 12-27

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.



Important: *The user-data portion of a smart large object does not pass through the physical-log buffers.*

The system page size is platform dependent. You can obtain the system page size through ON-Monitor under the Parameters:Shared-Memory option, which does not require the database server to be running, and also under the Parameters:Initialize option. The command **oncheck -pr**, which checks the root-dbspace reserved pages, also displays the system page size in the first section of its output.

PHYSDBS

<i>onconfig.std</i>	rootdb
<i>value</i>	
<i>if not present</i>	The dbspace name specified by ROOTNAME
<i>units</i>	A dbspace.
<i>range of values</i>	18 or fewer characters
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Physical-Log, Dbspace Name
<i>refer to</i>	“Where Is the Physical Log Located?” on page 24-8 and “Changing the Physical-Log Location and Size” on page 25-3

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

<i>onconfig.std</i>	1000
<i>value</i>	
<i>if not present</i>	200
<i>units</i>	kilobytes
<i>range of values</i>	Must be equal to or greater than 200 kilobytes
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Init, Phys. Log Size
<i>refer to</i>	“How Big Should the Physical Log Be?” on page 24-5 and “Changing the Physical-Log Location and Size” on page 25-3

PHYSFILE specifies the size of the physical log.

RA_PAGES

<i>onconfig.std</i>	none
<i>value</i>	
<i>if not present</i>	4 if MULTIPROCESSOR = 0, 8 if MULTIPROCESSOR = 1
<i>range of values</i>	RA_THRESHOLD < RA_PAGES < BUFFERS
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, perFormance, Num of Read Ahead Pages
<i>refer to</i>	“Configuring the Database Server to Read Ahead” on page 12-41

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 Chapter 2 of the [INFORMIX-Universal Server Performance Guide](#).

RA_THRESHOLD

onconfig.std	none
value	
if not present	RA_PAGES / 2
range of values	0 to (RA_PAGES-1)
takes effect	When shared memory is initialized
ON-Monitor	Parameters, perFormance, Read Ahead Threshold
refer to	“Configuring the Database Server to Read Ahead” on page 12-41

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.

If the value of RA_THRESHOLD is greater than the value of RA_PAGES, the **onconfig.std** value of RA_THRESHOLD is used instead. 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 Chapter 2 of the [INFORMIX-Universal Server Performance Guide](#).

RESIDENT

<i>onconfig.std</i>	0
<i>value</i>	
<i>range of values</i>	0=no; 1=yes
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Shared-Memory, Forced Residency
<i>utility</i>	onmode ; see page 39-39 .
<i>refer to</i>	“The Resident Portion of Database Server Shared Memory” on page 12-17

The RESIDENT parameter specifies whether the resident portion of Universal Server shared memory remains resident in operating-system physical memory. If your operating system supports forced residency, you can specify that the resident portion of shared memory not be swapped to disk.

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.

ROOTNAME

<i>onconfig.std</i>	rootdbs
<i>value</i>	
<i>range of values</i>	18 characters
<i>takes effect</i>	When disk is initialized (destroys all data)
<i>ON-Monitor</i>	Parameters, Initialize, Root Name
<i>refer to</i>	“Allocating Disk Space” on page 15-4

The ROOTNAME parameter specifies a name for the root dbspace for this Universal 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

<i>onconfig.std</i>	0
<i>value</i>	
<i>units</i>	kilobytes
<i>range of values</i>	Greater than or equal to 0
<i>takes effect</i>	When disk is initialized (destroys all data)
<i>ON-Monitor</i>	Parameters, Initialize, Offset (follows Root Size)
<i>refer to</i>	“Do You Need to Specify an Offset?” on page 15-6

ROOTOFFSET specifies the offset into the disk partition, device, or cooked file to reach the initial chunk of the root dbspace.

ROOTPATH

<i>onconfig.std</i>	/dev/online_root
<i>value</i>	
<i>range of values</i>	Any valid pathname
<i>takes effect</i>	When disk is initialized (destroys all data)
<i>ON-Monitor</i>	Parameters, Initialize, Primary Path
<i>refer to</i>	“Allocating Disk Space” on page 15-4

The ROOTPATH parameter 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**.

If you use raw disk space for your initial chunk, 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 Links to Each Raw Device” on page 15-7](#) presents a rationale for using links rather than device names.

ROOTSIZE

<i>onconfig.std</i>	20,000
<i>value</i>	
<i>if not present</i>	0
<i>units</i>	kilobytes
<i>range of values</i>	0 to maximum capacity of the storage device
<i>takes effect</i>	When disk is initialized (destroys all data)
<i>ON-Monitor</i>	Parameters, Initialize, Root Size
<i>refer to</i>	“Calculate the Size of the Root Dbspace” on page 14-33

ROOTSIZE specifies the size of the initial chunk of the root dbspace, expressed in kilobytes. If you decide to change the ROOTSIZE after you have initialized Universal Server, you must completely unload and reload your data.

SBSPACENAME

<i>onconfig.std</i>	none
<i>value</i>	
<i>if not present</i>	0
<i>range of values</i>	18 characters
<i>takes effect</i>	When disk is initialized
<i>ON-Monitor</i>	Parameters, Initialize, Root Size
<i>refer to</i>	“Create an Sbspace” on page 39-59

SBSPACENAME specifies the name of the system default sbspace. Unless you specify an sbspace name when you create a smart large object, Universal Server stores it in the default sbspace.

Even though you specify a default sbspace with the `SPSPACENAME` parameter, you must create the sbspace using the **-c -S** option of the **onspaces** utility before you can use it. Universal Server does not validate the name of the default sbspace until one of the following occurrences:

- You specify the default sbspace as the storage option for a CLOB or BLOB column in a `CREATE TABLE` statement
- Universal Server attempts to write a smart large object to the default sbspace because a no sbspace was specified for the column.

For information about sbspaces, refer to [“What Is an Sbspace?” on page 14-24](#). For information about assigning a smart large object to an sbspace, refer to the `CREATE TABLE` statement in the [Informix Guide to SQL: Syntax](#).

SERVERNUM

<i>onconfig.std</i>	0
<i>value</i>	
<i>range of values</i>	0 to 255
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Shared-Memory, Server Number
<i>refer to</i>	“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. Because the value 0 is included in the template file, **\$INFORMIXDIR/etc/onconfig.std**, Informix suggests that you choose a value other than 0 to avoid inadvertent duplication of SERVERNUM.

SHMADD

<i>onconfig.std</i>	8192
<i>value</i>	
<i>range of values</i>	1024 to 524,288
<i>units</i>	kilobytes
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Shared-Memory, Add Seg Size
<i>refer to</i>	“The Virtual Portion of Database Server Shared Memory” on page 12-28

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 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. The following command displays the size of memory segments and the amount of memory that is used or free in each:

```
onstat -g seg
```

Informix recommends the following rule of thumb for setting SHMADD initially:

- If physical memory is less than 256 megabytes, set SHMADD to 8 kilobytes.
- If physical memory is greater than 256 megabytes and less than 512 megabytes, set SHMADD to 16 kilobytes.
- If physical memory is greater than 512 megabytes, set SHMADD to 32 kilobytes.

SHMBASE

<i>onconfig.std</i> value	Platform dependent
<i>range of values</i>	Positive integers
<i>takes effect</i>	When shared memory is initialized
<i>refer to</i>	“UNIX Kernel Configuration Parameters” on page 13-4

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. ON-Monitor does not prompt for this value during initialization.

SHMTOTAL

<i>onconfig.std</i> value	0
<i>range of values</i>	integer greater than or equal to 1
<i>units</i>	kilobytes
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Shared-Memory, Total Memory
<i>refer to</i>	“How Much Shared Memory Does the Database Server Need?” on page 12-9

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

SHMTOTAL enables you to limit the demand for memory that the database server can place on your system. However, applications might fail if Universal 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
```

xx	current size of resident segments
yy	current size of virtual segments
zz	total shared-memory required.

SHMVIRTSIZE

onconfig.std	8000
<i>value</i>	
<i>if not present</i>	SHMADD, if present 8 kilobytes, if SHMADD is not present
<i>units</i>	kilobytes
<i>range of values</i>	Positive integers
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Shared-Memory, Non Res. Seg Size
<i>refer to</i>	“The Virtual Portion of Database Server Shared Memory” on page 12-28

SHMVIRTSIZE specifies the initial size of a virtual shared-memory segment. The basic algorithm for determining the size of the virtual portion of shared memory is as follows:

$$\text{fixed overhead} + \text{shared structures} + (\text{mncs} * \text{private structures})$$

<i>mncs</i> =	maximum number of concurrent sessions
<i>fixed overhead</i> =	global pool + thread pool after booting (partially dependent on the number of virtual processors).
<i>shared structures</i> =	AIO vectors + sort memory + dbspace backup buffers + dictionary size + size of stored procedure cache + histogram pool + other pools (see onstat display)
<i>private structures</i> =	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. Then use the following command to determine peak usage and lower the value of SHMVIRTSIZE accordingly:

```
% onstat -g seg
```

SINGLE_CPU_VP

onconfig.std	0
value	
range of values	0 = running with multiple CPU virtual processors 1 (or any other nonzero value) = running with only one CPU VP
takes effect	When shared memory is initialized
ON-Monitor	Parameters, perFormance, Single CPU VP
refer to	“Running on a Single-Processor Computer” on page 10-20

SINGLE_CPU_VP lets you specify 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 enables it to bypass many of the mutex calls that it must use when it runs multiple CPU virtual processors.



Important: The database server treats user-defined virtual processor classes as if they were CPU virtual processors. Thus, if you set SINGLE_CPU_VP to nonzero you cannot create any user-defined virtual processor classes.

If you set this parameter to nonzero and then attempt to bring up the database server with NUMCPUVPS or the VPCLASS cpu value for num= 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 this parameter to nonzero and then attempt to bring up the database server with a user-defined VPCLASS, you receive the following error message, and the database-server initialization fails:

```
oninit: Cannot have SINGLE_CPU_VP non-zero and user-defined VP classes
```

If you set SINGLE_CPU_VP to nonzero and attempt to add a CPU virtual processor, you receive the following message:

```
onmode: failed when trying to change the number of classname VPs 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

<i>onconfig.std</i>	32
<i>value</i>	
<i>units</i>	kilobytes
<i>range of values</i>	32 kilobytes to limit determined by Universal Server configuration and the amount of memory available
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Shared-Memory, Stack Size (kilobytes)
<i>refer to</i>	“Stacks” on page 10-15

The STACKSIZE parameter specifies the stack size for Universal 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 Universal 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.

User-defined routines are executed by user threads. You can increase the stack size for a particular routine using the **stack** modifier on the CREATE FUNCTION statement. For more information on the CREATE FUNCTION statement, refer to the [Informix Guide to SQL: Syntax](#).



Warning: Setting the value of STACKSIZE too low can cause stack overflow, the result of which is undefined but usually undesirable.

STAGEBLOB

<i>onconfig.std</i>	none
<i>value</i>	
<i>range of values</i>	18 or fewer characters
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Initialize, Stage Blob
<i>refer to</i>	INFORMIX-OnLine/Optical User Manual

Use this parameter only if you are storing simple large objects on optical storage with INFORMIX-OnLine / Optical. This parameter has no affect on ordinary blobspaces or sbspaces.

STAGEBLOB is the blobspace name for the area where INFORMIX-OnLine / Optical stages simple large objects that are destined for storage on optical disk.

TAPEBLK

<i>onconfig.std</i>	16
<i>value</i>	
<i>units</i>	kilobytes
<i>range of values</i>	Values greater than (page size)/ 1024
<i>takes effect</i>	The value is read from the ONCONFIG file by ontape when it executes. For onload and onunload , new values take effect when shared memory is initialized.
<i>ON-Monitor</i>	Parameters, Initialize, Block Size (follows Tape Dev)

The TAPEBLK parameter 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**.

Universal 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.

If you are using **onload** or **onunload**, you can specify a different block size on the command line. For information about **onload** and **onunload**, refer to the [Informix Migration Guide](#).

The system page size is platform dependent. You can obtain the system page size through ON-Monitor under the Parameters:Shared-Memory option, which does not require the database server to be running, and also under the Parameters:Initialize option. The command **oncheck -pr**, which checks the root-dbspace reserved pages, also displays the system page size in the first section of its output.

TAPEDEV

onconfig.std value	/dev/tapedev
if not present takes effect	/dev/null The value is read from the ONCONFIG file by ontape when it is executed. For onload and onunload , new values take effect when shared memory is initialized.
ON-Monitor	Parameters, Initialize, Tape Dev

The TAPEDEV parameter 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**.



Important: If you are using TAPEDEV to specify a device for **ontape** to use, read about setting and changing the value in the “[INFORMIX-Universal Server Archive and Backup Guide](#).” If you are using TAPEDEV to specify a device for **onunload** or **onload**, refer to the **onunload** or **unload** utility in the “[Informix Migration Guide](#).”

Using Symbolic Links

The value of TAPEDEV can be a symbolic link, enabling you to switch between tape devices without changing the pathname specified in TAPEDEV.

Using Remote Device

You can specify a tape device attached to another host computer using the following syntax:

```
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 the block size (“[TAPEBLK](#)” on [page 37-70](#)) and the tape size (“[TAPESIZE](#)” on [page 37-73](#)) are correct for the new device. If you need to change these values, you can do so at the same time that you change the value of TAPEDEV.

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. Universal 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. Universal 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	10240
<i>value</i>	
<i>units</i>	kilobytes
<i>range of values</i>	Positive integers
<i>takes effect</i>	The value is read from the ONCONFIG file by ontape when it is executed. For onload and onunload , new values take effect when shared memory is initialized.
<i>ON-Monitor</i>	Parameters, Initialize, Total Tape Size (follows Tape Dev)

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. For information about **onload** and **onunload**, refer to the [Informix Migration Guide](#).

TXTIMEOUT

onconfig.std	300
<i>value</i>	
<i>units</i>	Seconds
<i>range of values</i>	Positive integers
<i>takes effect</i>	When shared memory is initialized
<i>ON-Monitor</i>	Parameters, Shared-Memory, Transaction Timeout
<i>refer to</i>	“How the Two-Phase Commit Protocol Handles Failures” on page 34-10

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

<i>onconfig.std</i>	0
<i>value</i>	
<i>range of values</i>	0 = off; 1 = on
<i>takes effect</i>	during initialization
<i>ON-Monitor</i>	Parameters, perFormance, Use OS Time
<i>refer to</i>	<i>INFORMIX-Universal Server Performance Guide</i>

Setting USEOSTIME to 1 specifies that Universal 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.

VPCLASS

<i>syntax</i>	<i>classname, options</i> The <i>classname</i> is required. Unlike most configuration parameters, VPCLASS has several option fields that can appear in any order, separated by commas. You cannot use any white space in the fields. VPCLASS has the following options: <i>num=num_VPs</i> <i>max=max_VPs</i> <i>aff=affinity</i> <i>noage</i> <i>noyield</i> For more information about using these options, refer to the individual discussions later in this section.
<i>onconfig.std value</i>	none
<i>takes effect</i>	When shared memory is initialized.
<i>ON-Monitor</i>	None.
<i>refer to</i>	“User-Defined Classes of Virtual Processors” on page 10-24

The VPCLASS parameter allows you to designate a class of virtual processors and specify the following information for it:

- The number of virtual processors that the database server should start initially
- The maximum number of virtual processors allowed for this class
- The assignment of virtual processors to CPUs if processor affinity is available
- The disabling of priority aging by the operating system if the operating system implements priority aging

You can put several VPCLASS parameter definitions into your ONCONFIG file. Each VPCLASS parameter describes one class of virtual processors. Put each definition on a separate line, as in the following example:

```
VPCLASS cpu,num=8,aff=0-7,noage
VPCLASS new,num=0
```

Default Values for the VPCLASS Options

The following table shows the defaults and value ranges for the VPCLASS parameter options.

VPCLASS option	Class	Default Value	Range of Values
<i>num_VPs</i>	AIO	(2 * <i>number_of_chunks</i>) or 6, whichever is greater, where <i>number_of_chunks</i> is the number of chunks allocate	1 - 10,000
<i>num_VPs</i>	CPU	1 if MULTIPROCESSOR is 0 , 2 otherwise	1 - 10,000
<i>num_VPs</i>	All other classes	1	1 - 10,000
<i>max_VPs</i>	All	Unlimited	1 - 10,000
<i>affinity</i>	All	VPs are assigned to available processors in round-robin fashion.	Integers from 0 to (number of CPUs -1)
<i>noage</i>	All	Priority aging is in effect.	noage or omitted
<i>noyield</i>	User-defined	Threads will yield.	noyield or omitted

Interaction of VPCLASS With Other Configuration Parameters

Informix suggests that you use the VPCLASS parameter instead of the AFF_SPROC, AFF_NPROCS, NOAGE, NUMCPUVPS, and NUMAIOVPS parameters. If you use VPCLASS, you must explicitly remove other parameters from your ONCONFIG file. The following table shows which parameters you must remove.

If you use	You must remove
VPCLASS <code>cpu</code>	NUMCPUVPS, AFF_SPROC, AFF_NPROCS, NOAGE
VPCLASS <i>user-defined</i>	SINGLE_CPU_VP
VPCLASS <code>aio</code>	NUMAIOVPS

Naming the VPCLASS

The first item in the VPCLASS parameter gives the name of the virtual processor class that you are describing. The name can contain 18 alphanumeric characters and is not case sensitive. The name must be unique.

You can define new virtual processor classes for user-defined routines or DataBlade modules, or you can set values for a pre-defined virtual processor class. The following virtual processor classes are predefined by the database server and have specific functions:

<code>adm</code>	<code>lio</code>	<code>shm</code>
<code>adt</code>	<code>msc</code>	<code>soc</code>
<code>cpu</code>	<code>opt</code>	<code>str</code>
<code>kio</code>	<code>lio</code>	<code>tli</code>

The following example specifies that the database server should start three virtual processors of the CPU class:

```
VPCLASS cpu,num=3
```

Creating a User-Defined Class

You might want to describe a user-defined class of virtual processors to run DataBlade or user-defined routines. The following example creates the user-defined class **new**, for which the database server starts three virtual processors initially:

```
VPCLASS new,num=3
```

At a later time, you can use **onmode -p** to add virtual processors to the class. The following command adds three virtual processors to the **new** class:

```
onmode -p +3 new
```

For information about **onmode -p**, refer to [“Add or Remove Virtual Processors” on page 39-44](#).

When you create a user-defined routine or function, you assign it to a class of virtual processors using the CLASS parameter of the CREATE FUNCTION statement. When you create a class user-defined virtual processor to run user-defined routines, you must ensure that the name of the class agrees with the name that you assigned in the CREATE FUNCTION statement. For more information on how to assign a user-defined routine to either CPU or user-defined classes of virtual processors, refer to the CREATE FUNCTION statement, and specifically to the CLASS parameter, in the [Informix Guide to SQL: Syntax](#).



Tip: You can use the CREATE FUNCTION statement to create routines or functions that reference any user-defined class you like, and that class need not exist when the function is created. However, if you try to use a function that refers to a user-defined class, that class must exist and have virtual processors assigned to it. If the class does not have any virtual processors, you receive an SQL error. For information on running user-defined virtual processors with the **noyield** option, refer to [“Using the noyield Option” on page 37-80](#).

For more information on virtual processor classes, refer to [“Types of Virtual Processors” on page 10-7](#).

Using the num_VPs Option

The *num_VPs* option sets the number of virtual processors of the specified class that the database server should start during initialization.

On a single-processor computer, allocate only one CPU virtual processor. On a multiprocessor computer, do not allocate more CPU and user-defined virtual processors, combined, than there are CPUs on the computer.

Use the following syntax to specify the number of virtual processors:

```
num=num_VPs
```

For example, the following parameter specifies that the database server should start four virtual processors for the **cpu** class:

```
VPCLASS cpu,num=4
```

At a later time, you can use the **onmode -p** command to add virtual processors for the class. For information about **onmode -p**, refer to [“Add or Remove Virtual Processors” on page 39-44](#).

Using the max_VPs Option

The *max_VPs* option specifies the maximum number of virtual processors that the database server can start for the class.

Use the following syntax to specify the number of virtual processors:

```
max=max_VPs
```

The value can be any integer greater than 0. If you omit the *max_VPs* option, the number is unlimited.

Using the affinity Option

On multiprocessor computers that support *processor affinity*, the affinity option specifies the CPUs to which the database server binds virtual processors.

The affinity option has the following two forms:

```
aff=processor_number  
aff=start_range,end_range
```

In the first form, the database server binds all virtual processors in the class to the CPU numbered *processor_number*. (On a multiprocessor system, the operating system numbers the CPUs from 0 to (number of CPUs - 1)). In the second form, the database server assigns the virtual processors of the class to processors in the range *start_range* to *end_range*, inclusive. The value *end_range* must be larger than *start_range*, and all values must be less than the total number of available CPUs.

For example, if your platform has eight CPUs, your ONCONFIG file might include the following VPCLASS entries:

```
VPCLASS first,aff=3
VPCLASS second,num=3,aff=5-7
VPCLASS cpu,num=8,aff=0-7,noage
```

For more information about processor affinity, refer to [“Using Processor Affinity” on page 10-22](#).

Using the noyield Option

The **noyield** option causes a user-defined class of virtual processors to run user-defined routines in a way that gives the routine exclusive use of the virtual processor class. In other words, user-defined routines that use a **noyield** virtual processor class run serially, and each routine runs to completion before the next one begins. If the **noyield** option is set, you need not allocate more than one virtual processor for the class. A routine runs on a single virtual processor until it completes and additional virtual processors would be idle.

This option is ignored on predefined virtual processor classes such as CPU, AIO, and so on.

The following example specifies a user-defined class of virtual processors called **new**, which runs in no-yield mode:

```
VPCLASS new,noyield,num=1
```

For more information on specifying a user-defined class of virtual processors, refer to [“Creating a User-Defined Class” on page 37-78](#). For more information on the rationale behind the **noyield** option, refer to [“Specifying a Nonyielding User-Defined Virtual Processor” on page 10-25](#).

The sysmaster Database

What Is the sysmaster Database?	38-3
Using the System-Monitoring Interface	38-5
What Are the SMI Tables?	38-5
Accessing SMI Tables.	38-6
SELECT Statements	38-6
Triggers and Event Alarms	38-7
SPL and SMI Tables	38-7
Locking and SMI Tables	38-7
The System-Monitoring Interface Tables	38-8
sysadtinfo	38-9
sysaudit	38-10
syschkio	38-11
syschunks	38-11
sysconfig	38-13
sysdatabases.	38-14
sysdbslocale	38-15
sysdbspaces	38-15
sysdri	38-17
sysextents.	38-18
sysextspaces.	38-18
syslocks	38-19
syslogs.	38-20
sysprofile	38-21
sysptprof	38-23
sysseprof	38-24
syssessions	38-26
syssewts	38-28
systabnames.	38-29
sysvpprof.	38-30

The SMI Tables Map	38-30
Using SMI Tables to Obtain onstat Information	38-33

T

his 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 the ON-Archive tables, see the [*INFORMIX-Universal Server Archive and Backup Guide*](#).

What Is the sysmaster Database?

INFORMIX-Universal 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 Universal 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. These tables are described in the [INFORMIX-Universal Server Archive and Backup Guide](#).

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. Making changes could result in unpredictable and debilitating results.*

The database server creates the **sysmaster** database automatically when disk space is initialized. 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 **informix**, 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, if you wish to join **sysmaster** tables with tables in a nonlogging database, you must 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** that is stored in the **\$INFORMIXDIR/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 Universal Server operation.

What Are the SMI Tables?

The system-monitoring interface consists of a number 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 actually 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 allows you to 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 connect to databases managed by the database server and access or modify data.

Accessing SMI Tables

Any user can use SQL SELECT statements to query any SMI table except **sysadinfo** and **sysaudit**. Only user **informix** can query the SMI tables **sysadinfo** 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 a non-SMI table, you 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 would never be activated, even though the data returned by a SELECT statement would indicate that it should.

To create event alarms, you must 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 Universal 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

Universal Server supports the following SMI tables.

Table	Description	Reference
sysadinfo	Auditing configuration information	page 38-9
sysaudit	Auditing event masks	page 38-9
syschkio	Chunk I/O statistics	page 38-11
syschunks	Chunk information	page 38-11
sysconfig	Configuration information	page 38-13
sysdatabases	Database information	page 38-14
sysdblocale	Locale information	page 38-15
sysdbspaces	Dbpace information	page 38-18
sysdri	Data-replication information	page 38-17
sysextents	Extent-allocation information	page 38-18
sysextspaces	External spaces information	page 38-18
syslocks	Active locks information	page 38-20
syslogs	Logical-log file information	page 38-20
sysprofile	System-profile information	page 38-21
sysptprof	Table information	page 38-23
sysseprof	Counts of various user actions	page 38-24
syssessions	Description of each user connected	page 38-26
syssewts	User's wait time on each of several objects	page 38-28
systabnames	Database, owner, and table name for the tblspace	page 38-29
sysvpprof	User and system CPU used by each virtual processor	page 38-30

Many other tables in the **sysmaster** database are part of the system-monitoring interface but are not documented. Their schemas and column content might change from version to version.

sysadinfo

The **sysadinfo** table contains information about the auditing configuration for the database server. For more information, refer to the [INFORMIX-Universal Server Trusted Facility Manual](#). You must be user **informix** or **user** root to retrieve information from the **sysadinfo** table.

Column	Type	Description
adtmode	integer	If auditing is on or off n 0 off n 1 on
adterr	integer	Action on errors n 0 continually retry audit writes until they succeed. Processing for the thread that generated the error stops. n 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**, or for additional information about auditing, refer to the [INFORMIX-Universal Server 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
chknun	smallint	Chunk number
dbsnum	smallint	Dbspace number
nxchknun	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

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_sbchunk	integer	1 if the chunk is a sbpace, 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.

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 \$INFORMIXDIR/etc/\$ONCONFIG at boot time
cf_effective	char(256)	Value effectively in use
cf_default	char(256)	Value provided by Universal Server if no value is specified in \$INFORMIXDIR/etc/\$ONCONFIG

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
dbbs_dbsname	char(18)	Database name
dbbs_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
dbbsnum	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_sbspace	integer	1 if the dbspace is a sbpace, 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

Column	Type	Description
4	0x0004	dbspace mirroring disabled
8	0x0008	dbspace newly mirrored
16	0x0010	blob space
32	0x0020	blob space on removable media
64	0x0040	blob space on optical media
128	0x0080	blob space that has been dropped
256	0x0100	blob space 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 data-replication status of the database server.

Column	Type	Description
type	char(50)	Data replication type Possible values: primary secondary standard Not Initialized
state	char(50)	State of data replication Possible values: off on connecting failer read-only
name	char(20)	The name of the other database server in the data-replication pair
intvl	integer	The data-replication interval
timeout	integer	The 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

sysextspaces

The **sysextspaces** table provides information on external spaces. Indexes for the **id** column and the **name** column allow only unique values.

Column	Type	Description
id	integer	External space ID
name	char(18)	External space name
owner	char(8)	External space owner
flags	integer	External space flags (Reserved for future use)
refcnt	integer	External space reference count.
locsize	integer	Size of external space location, in bytes
location	char (256)	Location of external space

syslocks

The **syslocks** table provides information on all the currently active locks in the database server.

Column	Type	Description
dbname	char(18)	Database name
tablename	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
		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.

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		
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

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

(1 of 3)

Events Profiled in sysprofile	Description
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
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

(2 of 3)

Events Profiled in sysprofile	Description
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
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

(1 of 2)

Column	Type	Description
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
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)

sysstesprof

The **sysstesprof** 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

(1 of 2)

Column	Type	Description
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
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
		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	B-tree cleaner thread

(2 of 2)

syseswts

The **syseswts** 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: n unspecified n buffer n lock n asynchronous I/O n mt yield 0 n mt yield n n mt yield n checkpoint n log i/o n log copy n condition n lock mutex n lockfree mutex n deadlock mutex n lrus mutex n tblsp mutex n log mutex n ckpt mutex n mutex n mt ready n mt yield x n 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: n cpu n adm n lio n pio n aio n tli n soc n str n shm n opt n msc n adt
usercpu	float	Number of microseconds of user time
syscpu	float	Number of microseconds of system time

The SMI Tables Map

Figure 38-1 on page 38-31 displays the columns in the SMI tables.

Figure 38-1
Columns in the SMI tables

[illegible]

syseswts	sysesprof	syssessions	syslocks	sysextents	sysptprof	systabnames	sysdatabases
sid	sid	sid	dbsname	dbsname	dbsname	partnum	name
reason	lockreqs	username	tabname	tabname	tabname	dbsname	partnum
numwaits	locksheld	uid	rowidlk	start	partnum	owner	owner
cumtime	lockwts	pid	keynum	size	lockreqs	tabname	created
maxtime	deadlks	hostname	type		lockwts	collate	is_logging
	lktouts	tty	owner (sid)		deadlks		is_buff_log
	logrecs	connected	waiter (sid)		lktouts		is_ansi
	isreads	feprogram			isreads		is_nls
	iswrites	pooladdr			iswrites		flags
	isrewrites	is_wlatch			isrewrites		
	isdeletes	is_wlock			isdeletes		
	iscommits	is_wbuff			bufreads		
	isrollbacks	is_wckpt			bufwrites		
	longtxs	is_wlogbuf			seqscans		
	bufreads	is_wtrans			pagreads		
	bufwrites	is_monitor			pagwrites		
	seqscans	is_incrit					
	pagreads	state					
	pagwrites						
	total_sorts						
	dsksorts						
	max_sort diskspace						
	logspused						
	maxlogsp						

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, refer to [“onstat: Monitor Universal Server Operation” on page 39-75](#).

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)

Using SMI Tables to Obtain onstat Information

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 sysseprof	address wait nreads nwrites

(2 of 2)

Utilities

Using the -V Option	39-3
Multibyte Characters	39-4
oncheck: Check, Repair, or Display	39-5
ondblog: Change Logging Mode	39-25
oninit: Initialize Universal Server	39-27
onlog: Display Logical-Log Contents.	39-30
onmode: Mode and Shared-Memory Changes	39-35
onparams: Modify Log-Configuration Parameters	39-50
onspaces: Manage Blobspaces, Dbspaces, Extspaces, or Sbspaces	39-55
onstat: Monitor Universal Server Operation	39-75
ontape: Logging, Archives, and Restore.	39-107

This chapter provides reference material for the INFORMIX-Universal Server utilities. Universal Server utilities allow you to execute administrative tasks directly from your command line. The following utilities are documented in this chapter:

- **oncheck**
- **oninit**
- **onlog**
- **onmode**
- **onparams**
- **onspaces**
- **onstat**
- **ontape**

Except in the case of **oninit** and some **oncheck** and **onlog** options, you must initialize Universal Server before you execute any of the utilities.

Using the -V Option

All of the Informix command-line utilities allow you to 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.

Multibyte Characters

Universal Server utilities support multibyte command-line arguments. For a complete listing of the utilities that support multibyte command-line arguments, see Chapter 4 of the [Guide to GLS Functionality](#). ♦

oncheck: Check, Repair, or Display

Depending on the options 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 Universal Server manages, see [Chapter 42, “Disk Structures and Storage.”](#)

The **oncheck** utility is frequently confused with **onstat**. For a comparison of **oncheck** and **onstat**, see [“Comparing oncheck to onstat” on page 33-13.](#)

oncheck Check-and-Repair Options

The only structures that **oncheck** can repair are indexes. If **oncheck** detects inconsistencies in other structures, messages alert you to these inconsistencies, but **oncheck** cannot resolve the problem. For more details about Universal Server consistency checking, see [Chapter 31, “What Is Consistency Checking?”](#)

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 Recreate Index index_name for table_name
```

In addition, the **oncheck** utility does not repair external indexes accessed through a user-defined access method.

Any user can execute the check options. Only user **informix** or **root** can display database data or initiate index repairs.

What Does Each Option Do?

As shown in Figure 39-1, 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 39-1 associates **oncheck** options with their function.

Figure 39-1
oncheck Options and Their Function

Object	Check	Repair	Display
Blobspace simple large objects			-pB
Chunks and extents	-ce		-pe
Data rows, no simple large objects	-cd		-pd
Data rows, simple large objects, smart-large-object metadata	-cD		-pD
Index (key values)	-ci	-ci -y -pk -y	-pk
Index (keys plus rowids)	-cI	-cI -y -pK -y	-pK
Index (leaf key values)		-pl -y	-pl
Index (leaf keys plus rowids)		-pL -y	-pL
Metadata for smart large objects	-cs, -cS		-ps, -pS
Pages (by table or fragment)			-pp
Pages (by chunk)			-pP
Root reserved pages	-cr		-pr
Space usage (by table or fragment)			-pt

(1 of 2)

Object	Check	Repair	Display
Space usage (by table, with indexes)			-pT
System catalog tables	-cc		-pc
Table with a user-defined access method	-cd, -cD		
Index with a user-defined access method	-ci, -cI		

(2 of 2)

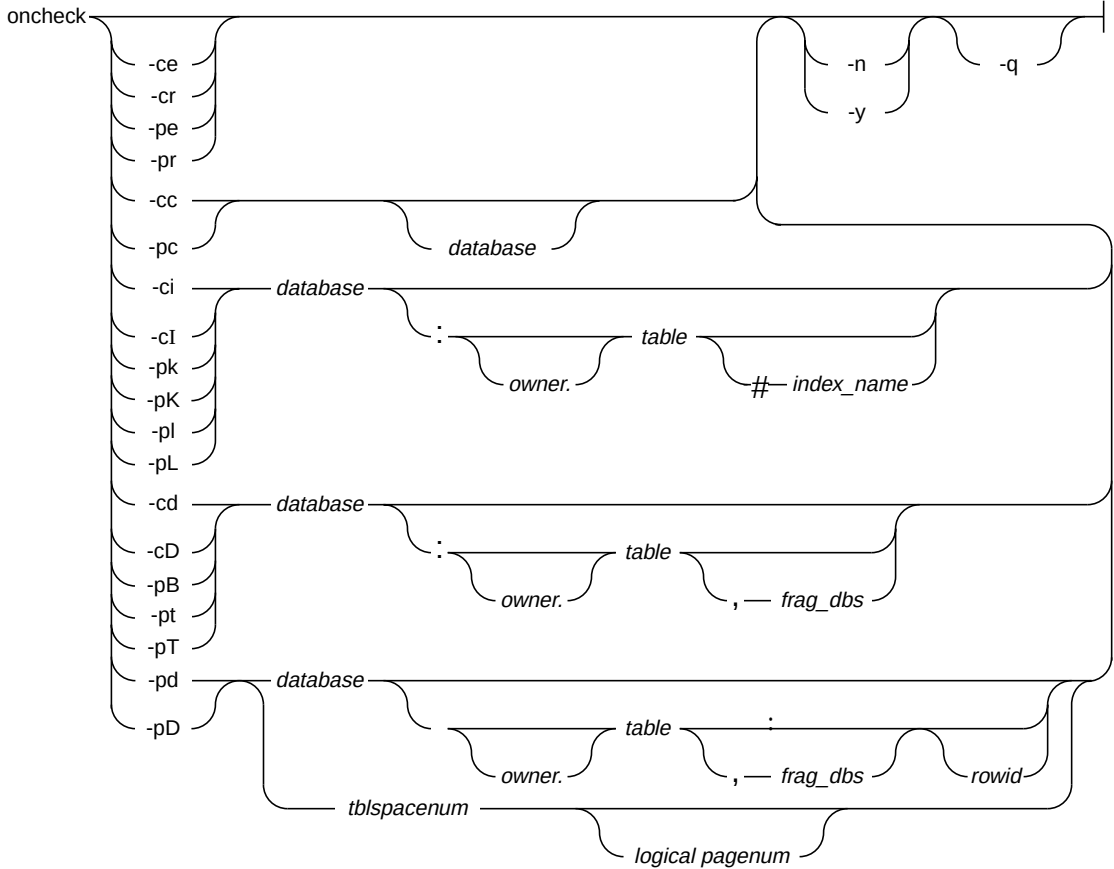
Locking and oncheck

The **oncheck** utility places a shared lock on tables when it checks 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 repairs an index.

When **oncheck** checks the consistency of an index, it locks the associated table in shared mode. However, in order for **oncheck** to repair an index, the utility must place an exclusive lock on the associated table.

Syntax

oncheck Options (1 of 2)



oncheck Options (2 of 2)

oncheck -pp database : table rowid
 owner. , — frag_dbs
 tblspace num — logical pagenum

-pP — chunk num — logical pagenum

-cs

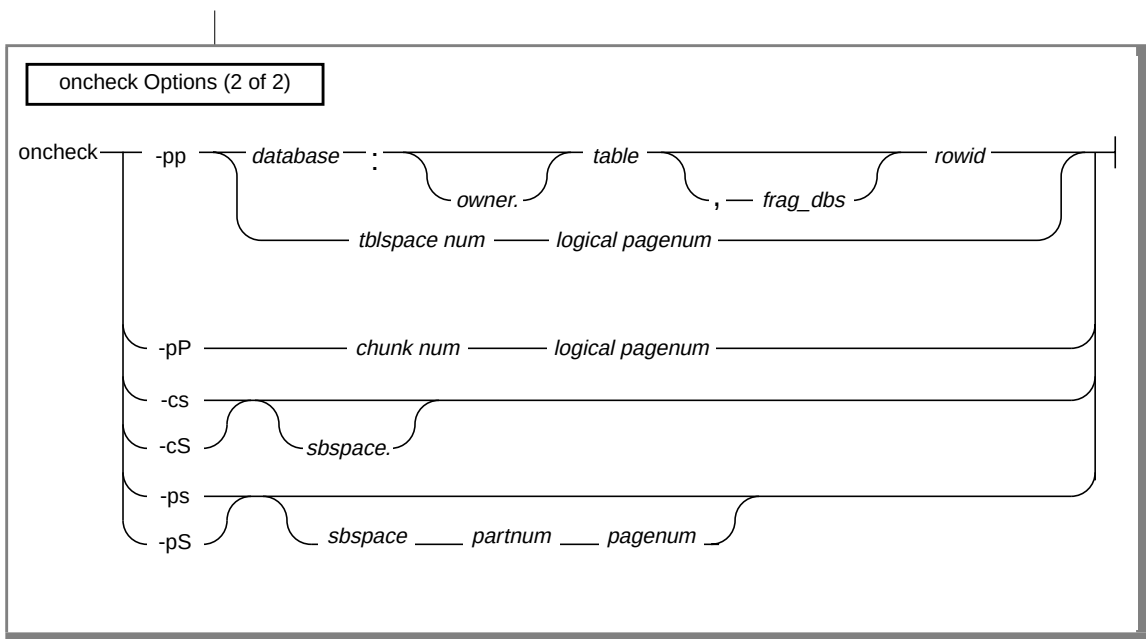
-cS sbspace.

-ps

-pS sbspace

-tr database
 : table
 owner. # — index_name

oncheck: Check, Repair, or Display



Element	Purpose	Key Considerations
-cc	Checks system catalog tables for the specified database. See “Using the -cc Option to Check System Catalog Tables” on page 39-17.	None.
-cd	Reads all pages except simple large objects from the tblspace for the specified database, table, or fragment and checks each page for consistency. Also checks tables employing a user-defined access method. See “Using the -cd and -cD Options to Check Pages” on page 39-17.	None.
-cD	Same as -cd but also reads the header of each blobpage and sbpage and checks it for consistency. See “Using the -cd and -cD Options to Check Pages” on page 39-17.	None.
-ce	Checks each chunk-free list and corresponding free space and each tblspace extent. Also checks smart-large-object extents and sbpace metadata. See “Using the -ce Option to Check the Chunk Free List” on page 39-19.	Additional Information: The oncheck process verifies that the extents on disk correspond to the current control information that describes them. References: For background information, see “Next-Extent Allocation” on page 42-31 and “Structure of the Chunk Free-List Page” on page 42-17, respectively.
-ci	Checks the ordering of key values and the consistency of horizontal and vertical node links for all indexes associated with the specified table. Also checks indexes employing a user-defined access method. See “Using the -ci and -cl Options to Check Index Node Links” on page 39-19.	None.

(1 of 6)

oncheck: Check, Repair, or Display

Element	Purpose	Key Considerations
-cI	Same as -ci but also checks that the key value tied to a rowid in an index is the same as the key value in the row. See “Using the -ci and -cI Options to Check Index Node Links” on page 39-19.	None.
-cs	Checks smart large object and sbspace metadata for a sbspace. See “Using the -cs, -cS, -ps, and -pS Options to Check and Display Smart Large Object and Sbspace Metadata” on page 39-21.	None.
-cS	Checks smart large object and sbspace metadata for a sbspace as well as extents. See “Using the -cs, -cS, -ps, and -pS Options to Check and Display Smart Large Object and Sbspace Metadata” on page 39-21.	None.
<i>sbspace</i>	Optional sbspace name. If not supplied, all sbspaces are checked.	None.
-cr	Checks each of the root dbspace reserved pages for several conditions. See “Using the -cr Option to Check Reserved Pages” on page 39-20.	None.
-n	Indicates that no index repair should be performed, even if errors are detected.	Additional Information: Used with the index repair options (-ci , -cI , -pk , -pK , -pl , and -pL).
-pB	Displays statistics that describe the average fullness of blob-space blobpages in a specified table. See “Using the -pB Option to Display Blobpage Statistics” on page 39-21.	Additional Information: These statistics provide a measure of storage efficiency for individual simple large objects in a database or table. If a table or fragment is not specified, statistics are displayed for the entire database. References: For background information, see “Optimizing Blobpage Size” on page 15-21.

(2 of 6)

Element	Purpose	Key Considerations
-pc	Same as -cc but also displays the system catalog information as it checks the system catalog tables, including extent use for each table.	None.
-pd	Displays rows in hexadecimal format. See “Using the -pd and -pD Options to Display Rows in Hexadecimal Format” on page 39-21.	None.
-pD	Displays rows in hexadecimal format and simple large object values stored in the tblspace or header information for simple large objects stored in a blobpage and smart large objects stored in a sbpage. See “Using the -pd and -pD Options to Display Rows in Hexadecimal Format” on page 39-21.	None.
-pe	Same as -ce but also displays the chunk and tblspace extent information as it checks the chunk free list and corresponding free space and each tblspace extent.	None.
-pk	Same as -ci but also displays the key values for all indexes on the specified table as it checks them. See “Using the -pk and -pK, -pl, and -pL Options to Display Index Information” on page 39-22.	None.
-pK	Same as -ci but also displays the key values and rowids as it checks them. See “Using the -pk and -pK, -pl, and -pL Options to Display Index Information” on page 39-22.	None.

(3 of 6)

oncheck: Check, Repair, or Display

Element	Purpose	Key Considerations
-pl	Same as -ci but also displays the key values, but only leaf-node index pages are checked. See “Using the -pk and -pK, -pl, and -pL Options to Display Index Information” on page 39-22.	None.
-pL	Same as -ci but also displays the key values and rowids, but only for leaf-node index pages. See “Using the -pk and -pK, -pl, and -pL Options to Display Index Information” on page 39-22.	None.
-pp	Displays contents of a logical page. See “Using the -pp and -pP Options to Display the Contents of a Logical Page” on page 39-23.	None.
-pP	Same as -pp but requires a chunk number and logical page number as input. See “Using the -pp and -pP Options to Display the Contents of a Logical Page” on page 39-23.	None.
-pr	Same as -cr but also displays the reserved-page information as it checks the reserved pages. See “Using -pr Option to Display Reserved-Page Information” on page 39-23.	None.
-ps	Checks and displays smart large object and sbspace metadata for a sbspace. See “Using the -cs, -cS, -ps, and -pS Options to Check and Display Smart Large Object and Sbspace Metadata” on page 39-21.	None.

(4 of 6)

Element	Purpose	Key Considerations
-pS	Checks and displays smart large object and sbspace metadata, as well as extents, for a sbspace. See “Using the -cs, -cS, -ps, and -pS Options to Check and Display Smart Large Object and Sbspace Metadata” on page 39-21.	None.
-pt	Displays tblspace information for a table or fragment. See “Using the -pt and -pT Options to Display Tblspace Information for a Particular Table or Fragment” on page 39-24.	None.
-pT	Same as -pt but also displays index-specific information and page-allocation information by page type (for dbspaces). See “Using the -pt and -pT Options to Display Tblspace Information for a Particular Table or Fragment” on page 39-24.	None.
-q	Suppresses all checking and validation messages.	None.
-tr	Frees empty pages in an R-tree index.	Additional information: The base table for each R-tree is exclusively locked while the index is compressed.
-y	Repairs indexes when errors are detected.	None.
<i>chunk num</i>	Specifies a decimal value that you use to indicate a particular chunk.	Restrictions: Value must be an unsigned integer greater than 0. Chunk must exist. Additional Information: Execute the -pe option to learn which chunk numbers are associated with specific dbspaces, blobspaces or sbspaces.
<i>database</i>	Specifies the name of a database that you wish to check for consistency.	Restrictions: Cannot check databases on remote database servers using oncheck . References: Syntax must conform to the Identifier segment; see Informix Guide to SQL: Syntax .

(5 of 6)

oncheck: Check, Repair, or Display

Element	Purpose	Key Considerations
<i>frag_dbs</i>	Specifies the name of a dbspace that contains a fragment you wish to check for consistency.	Restrictions: Dbspace must exist and contain the fragment that you wish to check for consistency. References: Syntax must conform to the Identifier segment; see Informix Guide to SQL: Syntax .
<i>index_name</i>	Specifies the name of the index that you wish to check for consistency.	Restrictions: Index must exist on table and in database specified. References: Syntax must conform to the Identifier segment; see Informix Guide to SQL: Syntax .
<i>logical page num</i>	Specifies an integer value that you use to indicate a particular page in a tblspace.	Restrictions: Value must be an unsigned integer between 0 and 16,777,215, inclusive. Additional Information: Can be expressed as an unsigned integer or hexadecimal that begins with 0x identifier. References: For background information, see “Logical Page Number” on page 42-38.
<i>owner</i>	Specifies the owner of a table.	Restrictions: Owner that you specify must be current owner of table. References: Syntax must conform to the Table Name segment; see Informix Guide to SQL: Syntax .
<i>pagenum</i>	Indicates the page number of the sbpace metadata portion to check and display.	None.
<i>partnum</i>	Identifies the sbpace metadata partition to check and display.	None.
<i>rowid</i>	Identifies the rowid of the row whose contents you wish to display. Rowid is displayed as part of oncheck -pD output.	Restrictions: Value must be an unsigned integer between 0 and 4,277,659,295, inclusive. Additional Information: Can be expressed as an unsigned integer or hexadecimal that begins with 0x identifier.
<i>sbspace</i>	Specifies the name of the sb-space that you wish to check for consistency.	None.
<i>table</i>	Specifies the name of the table that you wish to check for consistency.	Additional Information: Table exists when you execute the utility. References: Syntax must conform to the Table Name segment; see Informix Guide to SQL: Syntax .
<i>tblspace num</i>	Identifies the tblspace whose contents you wish to display.	Restrictions: Value must be an unsigned integer between 0 and 208,666,624, inclusive. Additional Information: Can be expressed as an unsigned integer or hexadecimal that begins with 0x identifier.

(6 of 6)

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 42-19](#).) 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 and sbpages, 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 42-24](#)) 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.)

If an index employing an access method provided by a DataBlade cannot find the access method, you receive the following message:

```
-9845 Access method access_method_name does not exist in  
database. Ensure that the DataBlade installation was  
successful.
```

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 and sbpages if any exist. The **-cD** option does not read entire blobpage or sbpage as it does with standard pages. Instead, **oncheck** reads only the header of each blobpage and sbpage 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 and sbpage. 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 and sbpage).

To monitor blobspace blobpages, refer to **oncheck -pB** ([“Using the -pB Option to Display Blobpage Statistics” on page 39-21](#)).

```
% oncheck -cD 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 42-31](#) and [“Structure of the Chunk Free-List Page” on page 42-17](#), respectively.) The **oncheck** process verifies that the extents on disk correspond to the current control information describing them. The -ce option also checks the smart large object extents and metadata information.

The **-pe** option performs the same checks and also displays the chunk and tblspace extent information as it checks it, as well as smart-large-object extent and metadata information.

```
% oncheck -ce
```

Using the -ci and -cl 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 42-48](#).)

If an index is not specified, all indexes are checked.

If a table is not specified, all tables in the database are checked.

When checking an external index, the user-defined access method determines what is checked. If an index that employs a user-defined access method cannot find the access method, an error is reported. The **oncheck** utility does not repair inconsistencies in external indexes.

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
```

The **oncheck** utility locks each table as its indexes are checked for both the **-ci** and **-ci** options.

Using the -cr Option to Check Reserved Pages

The **-cr** option checks each of the root dbspace reserved pages (see [“Reserved Pages” on page 42-6](#)) as follows:

- It validates the contents of the **\$INFORMIXDIR/etc/\$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.
- It checks all logical-log and physical-log pages for consistency.

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** detects the inconsistency and returns an error message.

The **-pr** option performs the same checks and also displays the reserved-page information as it checks the reserved pages.

```
% oncheck -cr
```

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 -cs, -cS, -ps, and -pS Options to Check and Display Smart Large Object and Sbspace Metadata

The **-cs** and **-cS** options validate metadata for a sbspace. If the sbspace name is not specified, all sbspaces are checked. The following example checks the sbspace, test_sbspace:

```
% oncheck -cs test_sbspace
```

The **-ps** and **-pS** options validate and display metadata for a sbspace. If the sbspace name is not specified, all sbspaces are checked. Metadata partition and page number can be optionally specified. The following example checks and displays metadata on sbpage 32 in partition 25:

```
% oncheck -ps test_sbspace 25 32
```

Using the -pB Option to Display Blobpace 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 simple large 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 15-21.](#))

```
% 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) 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. Simple-large-object descriptors stored in the data row are printed, but simple-large-object data itself is not.

The **-pD** option prints the same information as **-pd**. In addition, **-pD** prints simple-large-object values stored in the tblspace or header information for simple large objects stored in a blob space blobpage.

```
% oncheck -pd stores7:customer,frgmt1
% 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 -cl Options to Check Index Node Links” on page 39-19.](#)) 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 -cl Options to Check Index Node Links” on page 39-19.](#)) 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 42-48.](#)

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 -pK -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 you wish 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_dbspc1 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 Option to Check Reserved Pages” on page 39-20](#).) 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 42-6](#).

```
% 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 the table was created. The Extents fields list the physical address for the 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 the [INFORMIX-Universal Server 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 the -tr Option To Compress an R-tree Index

The **-tr** option frees empty pages in an R-tree index. You must specify the name of the database and you can also, optionally, specify the owner, the name of a table, and the index. The following example illustrates the **-tr** option.

```
oncheck -tr mydatabase:mytable
```

This command compresses all R-tree indexes for **mytable** in the **mydatabase** database.

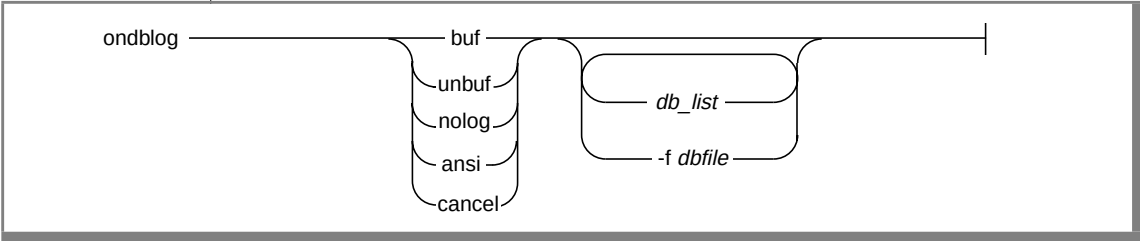
Each table for each R-tree index that **oncheck** compresses will be locked exclusively while **oncheck** performs the compression. For more information on R-tree indexes, refer to [Chapter 42, “Disk Structures and Storage”](#).

ondblog: Change Logging Mode

The **ondblog** utility allows you to 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 transaction information is written to a buffer before it is written to a logical log.	None.
unbuf	Sets the logging so 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.

Element	Purpose	Key Considerations
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 <i>dbfile</i> .	Additional Information: This command is useful if the list of databases is very long or often used.
<i>db_list</i>	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 server are modified.

(2 of 2)

These **ondblog** command options are similar to those available from the Database option of the ON-Monitor **Logical-Logs** menu.

For information on how to change the logmode using ON-Archive, see Chapter 8 of the [INFORMIX-Universal Server Archive and Backup Guide](#).

For considerations about changing the logging status of a database, see [Chapter 21, “Managing Database-Logging Status.”](#)

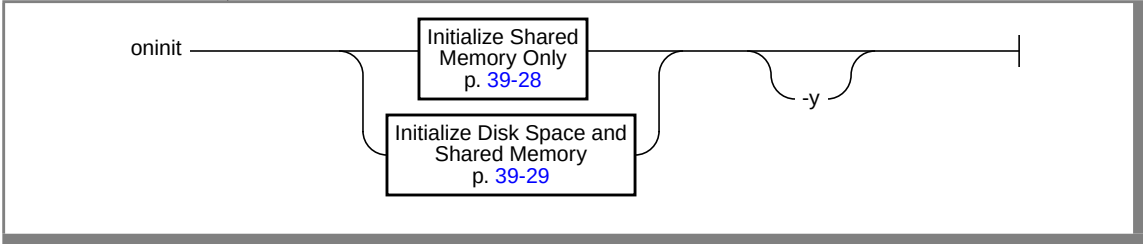
oninit: Initialize Universal Server

Executing **oninit** from the command line initializes Universal Server shared memory and brings Universal Server on-line.

If you use options to **oninit**, you can also initialize disk space. You must be logged in as **root** or user **informix** to execute **oninit**.

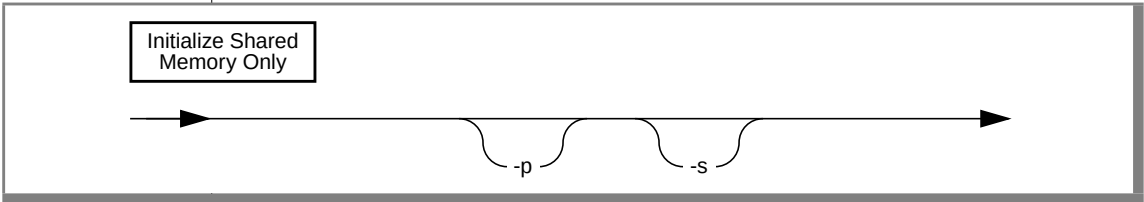
For information about what happens during initialization, refer to [Chapter 9, “What Is Initialization?”](#)

Syntax



Element	Purpose	Key Considerations
-y	Causes Universal 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, Universal 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 Universal Server in quiescent mode. See “Initializing Shared Memory Using the -s Option” on this page.	Additional Information: This option is equivalent to the ON-Monitor Mode menu, Startup option. Universal Server should be in off-line mode to initialize shared memory.

Initializing Shared Memory Using No Options

If you execute **oninit** without options, Universal Server is left in on-line mode after shared memory is initialized. For example, the following commands take Universal Server off-line and then reinitialize shared memory:

```
% onmode -ky
% oninit
```

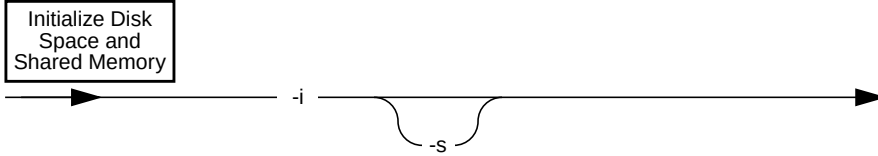
Initializing Shared Memory Using the -s Option

The **-s** option initializes shared memory and leaves Universal Server in quiescent mode. This option is equivalent to the ON-Monitor Mode menu, Startup option.

The following commands take Universal Server off-line, then reinitialize shared memory, and leave Universal Server in quiescent mode:

```
% onmode -ky
% oninit -s
```

Initialize Disk Space and Shared Memory



Element	Purpose	Key Considerations
-i	Causes Universal Server to initialize disk space and shared memory. Leaves Universal Server in on-line mode after initializing disk space.	None.
-s	When used with -i , the -s option causes Universal Server to be left in quiescent mode after disk initialization.	None.



Warning: When you initialize disk space, the initialization destroys all data that your Universal Server database server currently manages.

Universal Server must be off-line when you initialize disk space.

Specify the Number of Virtual Processors

The VPCLASS configuration parameter allows you to specify, for each class of virtual processors, the number of VPs that the database server should start on initialization. Alternatively, you can use NUMCPUVPS and NUMAIOVPS to specify the initial number of VPs for the CPU and AIO classes. However, you cannot use both VPCLASS and NUMCPUVPS and NUMAIOVPS in the same configuration file. If your ONCONFIG file contains conflicting parameters, **oninit** returns one of the following messages:

```
oninit: Can't mix VPCLASS cpu and NUMCPUVPS, SINGLE_CPU_VP,
AFF_SPROC, AFF_NPROCS, or NOAGE parameters
```

```
oninit: Can't mix VPCLASS aio and NUMAIOVPS parameters
```

For more information about VPCLASS, refer to [“VPCLASS” on page 37-75](#).

onlog: Display Logical-Log Contents

The **onlog** utility displays the contents of a Universal Server 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, described in the [INFORMIX-Universal Server Archive and Backup Guide](#).)

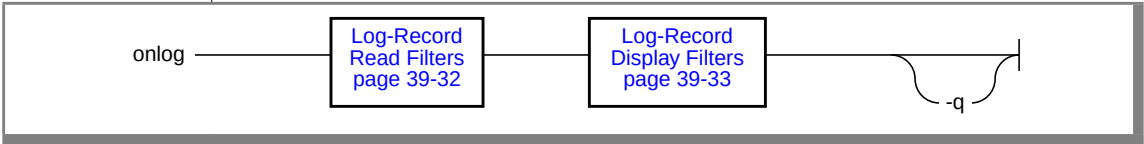
The **onlog** output is most useful in debugging situations, when you want to track a specific transaction or see what changes have been made to a specific tblspace. (For help in interpreting the logical-log file contents, see [Chapter 41, “Interpreting Logical-Log Records.”](#))

Any user can run all of the **onlog** options except the **-l** option. Only user **informix** can run the **-l** option.

If Universal Server is in off-line mode when you execute **onlog**, only the files on disk are read. If Universal 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 Universal Server reads a log file with status U (see the **-l** option of the **onstat** utility described on [page 39-93](#)) from disk while in on-line mode, Universal 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

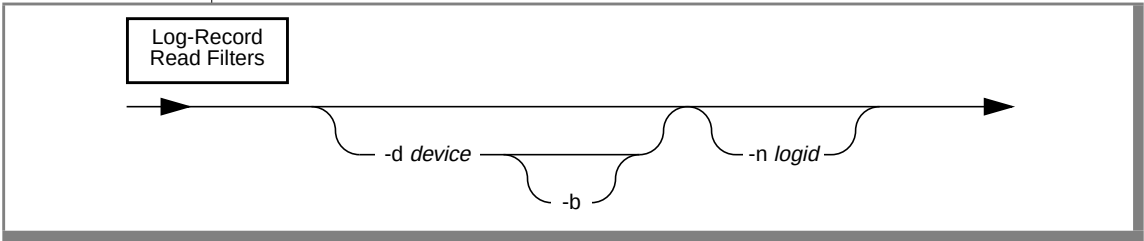
You can display every logical-log record header, or you can specify output based on the following criteria:

- 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.

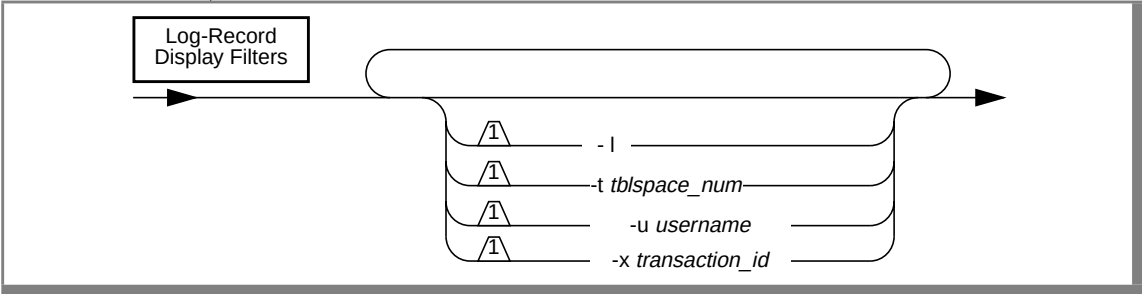
Log-Record Read Filters



Element	Purpose	Key Considerations
-b	Displays logical-log records associated with blob space blob pages.	Additional Information: Universal Server stores these records on the logical-log backup tape as part of blob space logging.
-d device	Names the pathname of the tape device where the logical-log backup tape whose contents you wish to display is mounted.	Additional Information: The device 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 you specify using <i>logid</i> .	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 background information on the onstat utility, see page 33-17 .

The **onlog** utility uses the pathnames that are stored in the root db space 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 <i>tblspace_num</i>	Displays records associated with the <i>tblspace</i> 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 <i>tblspace</i> number of a particular <i>tblspace</i> , query the systables system catalog table as described on page 42-20 .
-u <i>username</i>	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 <i>transaction_id</i>	Displays only records associated with the transaction 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 occurs, Universal 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 specify no 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 **-u** and **-x** options, only the activities initiated by the specified user during the specified transaction are displayed. If you use both the **-u** and **-t** options, only the activities initiated by the specified user and associated with the specified tblspace are displayed.

onmode: Mode and Shared-Memory Changes

The flags that accompany **onmode** determine which of the following operations is performed:

- Change Universal Server operating mode
- Force a checkpoint
- Change residency of Universal Server resident shared memory
- Switch the logical-log file
- Kill a Universal Server database server session
- Kill a Universal 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 INFORMIX-OnLine Dynamic Server Version 7.2 format (to revert to that product release). For more information on migrating from or reverting to INFORMIX-OnLine Dynamic Server, refer to the [Informix Migration Guide](#).
- Regenerate **.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**.

Syntax

onmode

-y

[Change Universal Server Modes, page 39-37](#)

[Force a Checkpoint, page 39-39](#)

[Change Shared-Memory Residency, page 39-39](#)

[Switch the Logical-Log File, page 39-40](#)

[Kill a Universal Server Session, page 39-40](#)

[Kill a Universal Server Transaction, page 39-41](#)

[Set Data-Replication Types, page 39-42](#)

[Add a Shared-Memory Segment, page 39-43](#)

[Add or Remove Virtual Processors, page 39-44](#)

Change Database Format with
onmode -b
see Informix Migration Guide

[Regenerate .infos File, page 39-45](#)

[Change Decision-Support Parameters, page 39-46](#)

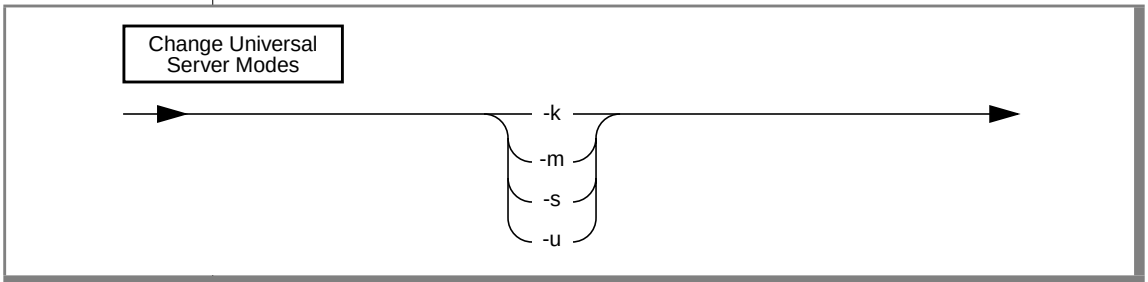
[Free Unused Memory Segments, page 39-47](#)

[Override ONDBSPDOWN WAIT Mode, page 39-48](#)

Element	Purpose	Key Considerations
-y	Causes Universal Server to automatically respond “yes” to all prompts.	None.

If no options are used, Universal Server returns a usage statement.

Change Universal Server Modes



Element	Purpose	Key Considerations
-k	Takes Universal Server to off-line mode and removes Universal Server shared memory. See “Taking Universal Server to Off-Line Mode Using the -k Option” on page 39-38.	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.
-m	Takes Universal Server from quiescent to on-line mode.	Additional Information: The -m option is equivalent to the ON-Monitor On-Line option.
-s	Shuts down Universal Server gracefully. See “Shutting Down Universal Server Gracefully with the -s Option” on page 39-38.	Additional Information: The -s option is equivalent to the ON-Monitor Graceful-Shutdown option. Users who are already using Universal Server are allowed to finish before Universal Server comes to quiescent mode, but no new connections are allowed. When all processing is finished, -s takes Universal Server to quiescent mode. The -s option leaves shared memory intact.
-u	Shuts down Universal Server immediately. See “Shutting Universal Server Down Immediately with the -u Option” on page 39-38.	Additional Information: The -u option is equivalent to the ON-Monitor Immediate-Shutdown option. This option brings Universal 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 Universal Server from one mode to another mode.

Taking Universal Server to Off-Line Mode Using the -k Option

This option takes Universal Server to off-line mode and removes Universal 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 Universal Server comes off-line. If you want to eliminate these prompts, execute the **-y** option with the **-s** option.

Shutting Down Universal Server Gracefully with the -s Option

The **-s** option is equivalent to the ON-Monitor Graceful-Shutdown option. Users who are already using Universal Server are allowed to finish before Universal Server comes to quiescent mode, but no new connections are allowed. When all processing is finished, **-s** takes Universal Server to quiescent mode. The **-s** option leaves shared memory intact.

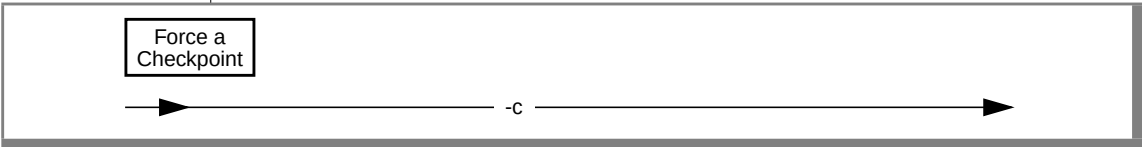
A prompt asks for confirmation. If you want to eliminate this prompt, execute the **-y** option with the **-s** option.

Shutting Universal Server Down Immediately with the -u Option

The **-u** option is equivalent to the ON-Monitor Immediate-Shutdown option. This option brings Universal Server to quiescent mode without waiting for users to finish their sessions. Their current transactions are rolled back, and their sessions are terminated.

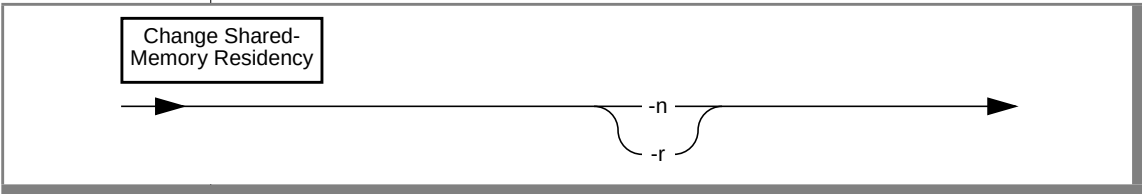
A prompt asks for confirmation. Another prompt asks for confirmation to kill user threads before Universal Server comes to quiescent mode. If you want to eliminate these prompts, execute the **-y** option with the **-s** option.

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 might 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 Universal Server 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 Universal Server shared memory.	Additional Information: This command does not affect the value of RESIDENT, the forced-memory parameter in the ONCONFIG file.

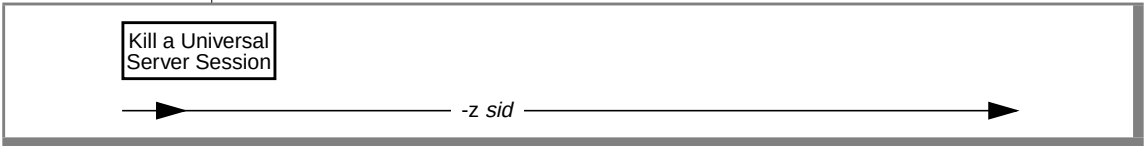
To change the forced-residency setting in the ONCONFIG configuration file, see [“Turning Residency On or Off for the Next Time You Reinitialize Shared Memory”](#) on page 13-15.

Switch the Logical-Log File



Element	Purpose	Key Considerations
-l	Switches the current logical-log file to the next logical-log file.	Additional Information: You must use onmode to switch to the next logical-log file. ON-Monitor has no equivalent option. References: For background information, see “Switching to the Next Logical-Log File” on page 23-15.

Kill a Universal Server Session



Element	Purpose	Key Considerations
-z sid	Kills the session that you specify in <i>sid</i> .	Restrictions: 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**, Universal Server attempts to kill the specified session. If Universal Server is successful, it frees any resources held by the session. If Universal 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 Universal Server off-line, as described in [“Taking Universal Server to Off-Line Mode Using the -k Option” on page 39-38](#), to close all sessions.

Kill a Universal Server Transaction

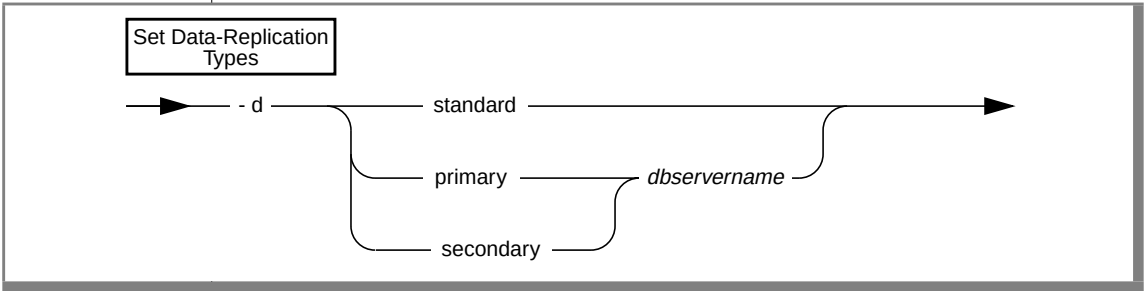


Element	Purpose	Key Considerations
-Z address	Kills a distributed transaction associated with the shared-memory address <i>address</i> .	Restrictions: This argument must be the address of an ongoing distributed transaction that has exceeded the amount of time specified by TXTIMEOUT. Address must conform 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 Universal Server database server involved with another Universal Server database server in distributed transactions. References: For background information, see “Independent Actions” on page 34-18.



Warning: *If applications are performing distributed transactions, killing one of the distributed transactions can leave your client/server database system in an inconsistent state. Avoid this situation, if possible.*

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 Universal Server is in quiescent mode. You can use the -d standard option when Universal Server is in quiescent, on-line, or read-only mode.
<i>dbservername</i>	Identifies the database server name of the primary or secondary database server.	Restrictions: The <i>dbservername</i> argument must correspond to the DBSERVERNAME parameter in the ONCONFIG file of the intended secondary database server. It should <i>not</i> correspond to one of the servers specified by the DBSERVERALIASES parameter. Additional Information: The <i>dbservername</i> 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 background information, see <i>range of values</i> for the DBSERVERNAME configuration parameter on page 37-15 .

-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.

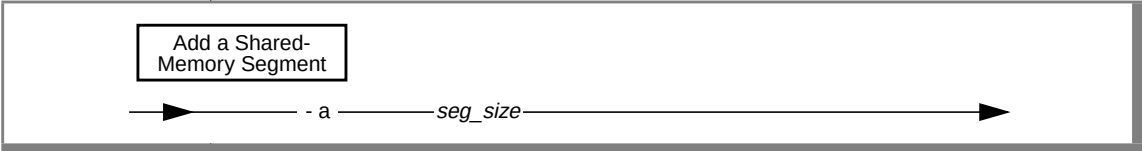
-d primary dbservername

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

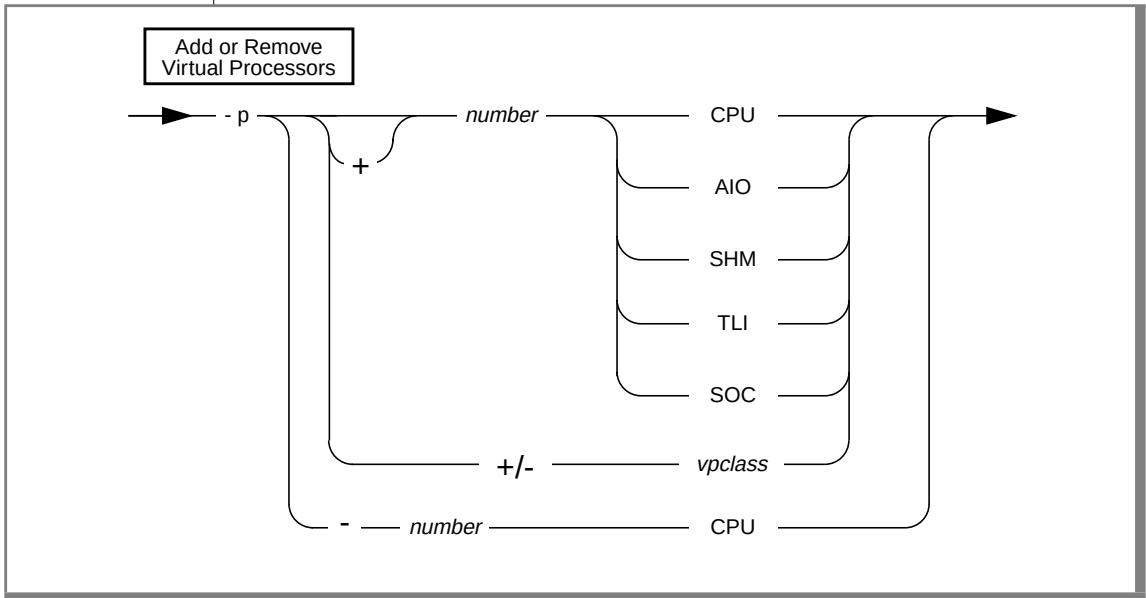


Element	Purpose	Key Considerations
-a <i>seg_size</i>	Allows you to add a new virtual shared-memory segment. Size is specified in kilobytes.	Restrictions: Value of <i>seg_size</i> must be a positive integer. It must not exceed the operating-system limit on the number of shared-memory segments.

Ordinarily, you do not need to add segments to the virtual portion of shared memory because Universal Server automatically adds segments as they are needed. However, as segments are added, Universal 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 Universal Server exhausts the number of available segments before it acquires the memory that it needs for some operation.

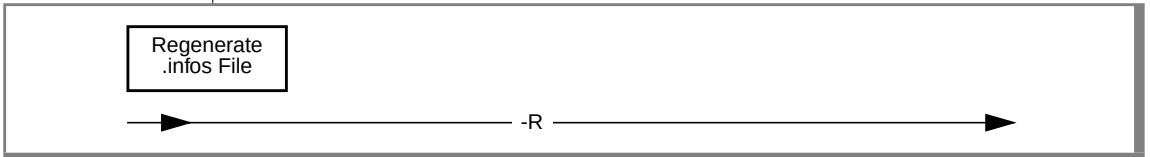
If you manually add a segment that is larger than the segment specified by SHMADD, you can avoid exhausting the operating-system limit for segments but still meet the need that Universal Server has for additional memory.

Add or Remove Virtual Processors



Element	Purpose	Key Considerations
-p <i>number</i>	Add or remove virtual processors. The number of virtual processors to add or remove is specified by <i>number</i> .	Restrictions: Value must be an unsigned integer greater than 0. You can use the -p option only when Universal 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.
<i>vpclass</i>	Names an extension class.	Additional Information: The extension class is defined by the VPCLASS parameter in the ONCONFIG file. For more information on extension classes, see “VPCLASS” on page 37-75 .

Regenerate .infos File

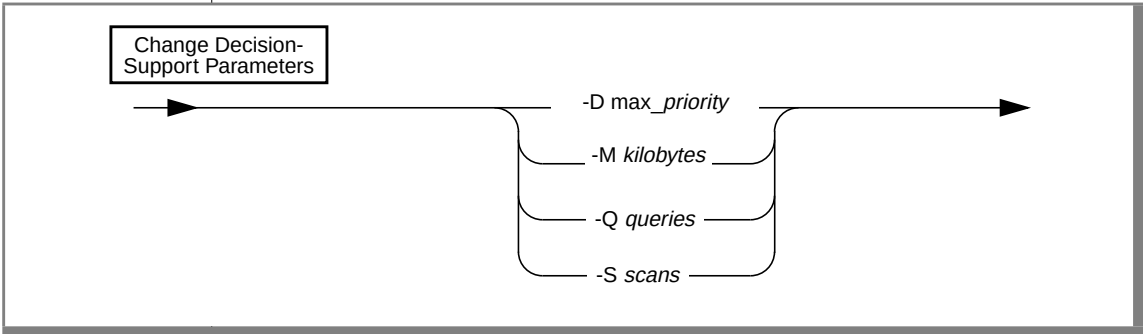


Element	Purpose	Key Considerations
-R	Re-creates the .info.dbservername file.	Restrictions: Before you use the -R option, you must set the \$INFORMIXSERVER environment variable to match the DBSERVERNAME parameter from the ONCONFIG file. Do not use the -R option with \$INFORMIXSERVER set to one of the DBSERVERALIAS names.

When Universal Server accesses utilities, it uses information from the **\$INFORMIXDIR/etc/.infos.dbservername** file. Universal Server creates and manages this file, and you should never need to do anything to the file.

However, if the **.infos.dbservername** file 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: Value must be an unsigned integer between 0 and 100. Additional Information: You specify <i>max_priority</i> as a factor to temper the user's request for PDQ resources. References: For background information, see “Limiting the Priority of PDQ Queries” on page 19-4 .
-M kilobytes	Changes the value of DS_TOTAL_MEMORY.	Restrictions: Value must be an unsigned integer between 128 * DS_MAX_QUERIES and 1,048,576. Additional Information: You specify <i>kilobytes</i> for the maximum amount of memory available for parallel queries. References: For background information, see “Adjusting the Amount of Memory” on page 19-7 .
-Q queries	Changes the value of DS_MAX_QUERIES.	Restrictions: Value must be an unsigned integer between 1 and 8,388,608. Additional Information: You specify <i>queries</i> for the maximum number of concurrently executing parallel queries. References: For background information, see “Limiting the Maximum Number of Queries” on page 19-9 .

(1 of 2)

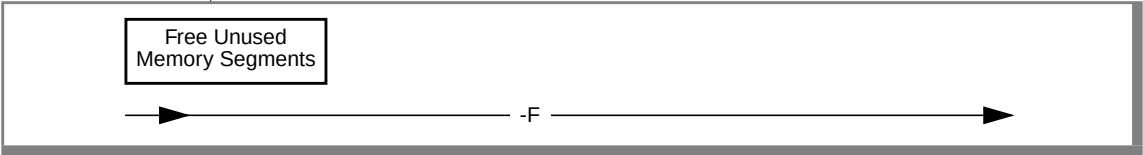
Element	Purpose	Key Considerations
-S scans	Changes the value of DS_MAX_SCANS.	Restrictions: Value must be an unsigned integer between 10 and 1,048,576. Additional Information: You specify <i>scans</i> for the maximum number of concurrently executing parallel scans. References: For background information, see “Limiting the Number of Concurrent Scans” on page 19-8.

(2 of 2)

These options allow you to change Universal Server configuration parameters while Universal Server is on-line. The new values affect only the current instance of Universal 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, refer to [Chapter 37, “Configuration Parameters.”](#)

Free Unused Memory Segments



Element	Purpose	Key Considerations
-F	Frees unused memory segments.	None.

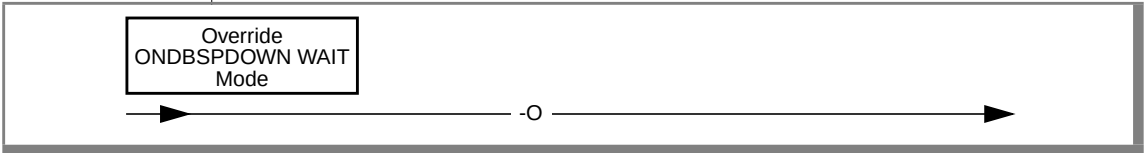
When you execute **onmode -F**, the Universal Server memory manager examines each memory pool for unused memory. When the memory manager locates 8 kilobytes of contiguous unused memory, it is immediately freed. After the memory manager checks each memory pool, it begins checking memory segments and frees any that Universal Server no longer needs.

Informix recommends that you run **onmode -F** on a regular basis from **cron** or some other operating-system scheduling facility. In addition, consider running this **onmode** option after you direct Universal Server to perform some function that creates additional memory segments such as large index builds, sorts, or backups.

Running the **-F** option causes a significant degradation of performance for any users that are active when you execute the utility. Although brief (1 to 2 seconds), degradation for single-user systems 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 Universal Server to block all updating threads.
- You cannot or do not wish to correct the problem that caused the disabling I/O error.
- You want Universal Server to mark the disabled dbspace as down and continue processing.

When you execute this option, Universal 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.
```

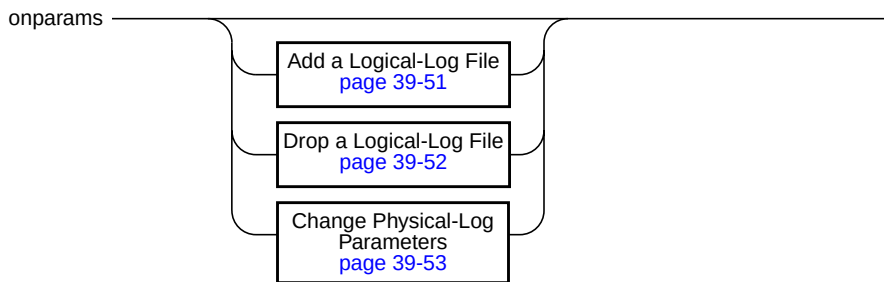
onparams: Modify Log-Configuration Parameters

The flags that accompany **onparams** determine which of the following operations is performed:

- Add a logical-log file
- Drop a logical-log file
- Change the size or location of the physical log

Universal Server must be in quiescent mode, and you must be logged in as user **root** or user **informix** to execute **onparams**.

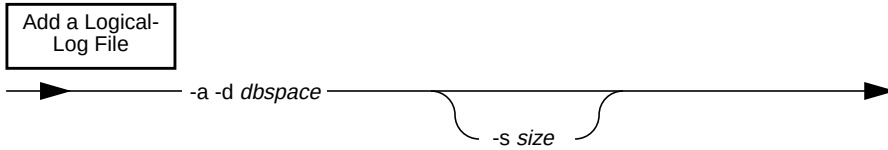
Syntax



Any **onparams** command fails if a Universal Server dbspace backup is in progress.

If no options are used, **onparams** returns a usage statement.

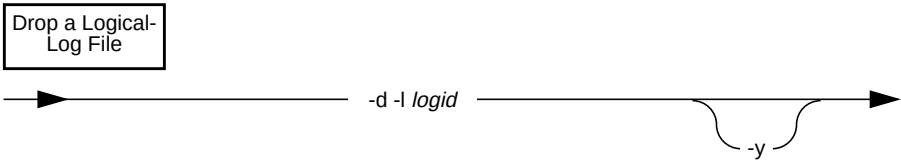
Add a Logical-Log File



Element	Purpose	Key Considerations
-a -d dbspace	Adds a logical-log file in the location specified by <i>dbspace</i> .	Additional Information: The space allocated for a logical-log file must be contiguous. Universal 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 archive. If you are using ON-Archive, you only need to create a level-0 archive 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: 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 Universal Server disk space was initialized. References: For background information, see “Changing LOGSIZE or LOGFILES” on page 23-10.

Using the **-a** option of **onparams** to add 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 Dbpace” on page 23-8](#).

Drop a Logical-Log File

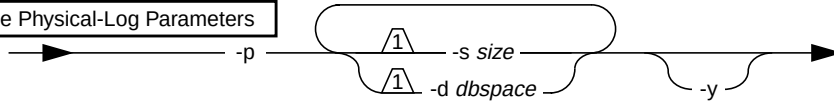


Element	Purpose	Key Considerations
-d -l logid	Allows you to drop a logical-log file specified by <i>logid</i> .	Restrictions: Value must be an unsigned integer greater than or equal to 0. Additional Information: You can obtain the <i>logid</i> from the number field of onstat -l . Universal Server requires a minimum of three logical-log files at all times. You cannot drop a log file if Universal Server is configured for three logical-log files. Universal 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 status of Free (F) or newly Added (A). After your configuration reflects the desired number of log files, create a level-0 archive. If you are using ON-Archive, you only need to create a level-0 archive of the root dbspace. Use onstat -l to view the status of your logical-log files.
-y	Causes Universal Server to automatically respond “yes” to all prompts.	None.

The **onparams** command to drop 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 23-8](#).

Change Physical-Log Size or Location

Change Physical-Log Parameters



Element	Purpose	Key Considerations
-d dbspace	Changes the location of the physical log to the specified <i>dbspace</i> .	Additional Information: The space allocated for the physical log must be contiguous. References: Syntax must conform to the Identifier segment; see <i>Informix Guide to SQL: Syntax</i> .
-p	Changes the location or size of the physical log.	None.
-s size	Changes the size (in kilobytes) of the physical log.	Restrictions: Value must be an unsigned integer greater than or equal to 200 kilobytes. Additional Information: If you move the log to a <i>dbspace</i> 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 Universal 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 Universal Server recovery. 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 additional information on this topic, see [“Changing the Physical-Log Location and Size” on page 25-3](#).

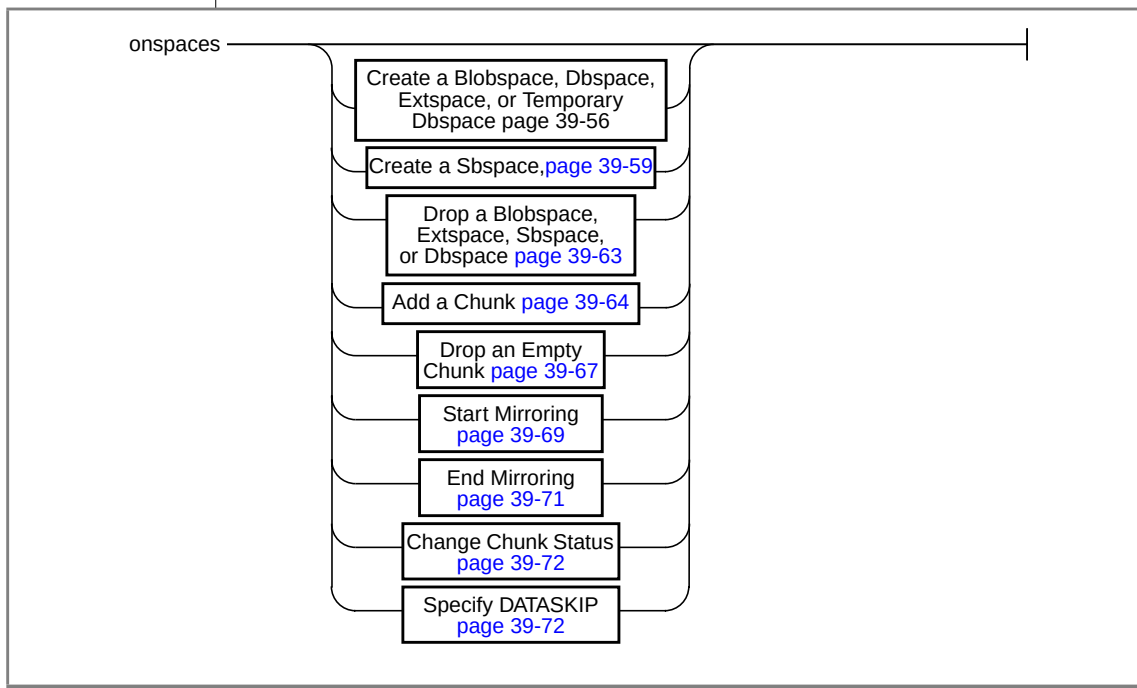
onspaces: Manage Blobspaces, Dbspaces, Extspaces, or Sbspaces

The **onspaces** utility enables you to perform the following tasks:

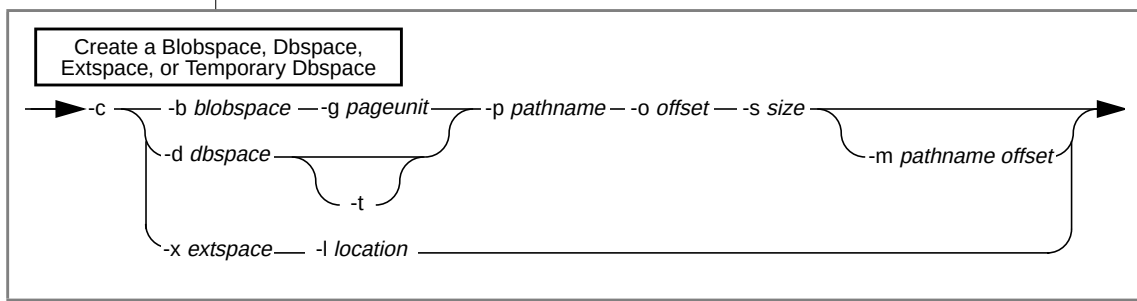
- Create a blobspace, extspace, sbspace, dbspace, or temporary dbspace
- Drop a blobspace, extspace, sbspace, or dbspace
- Add a chunk
- Drop a chunk
- Start mirroring
- End mirroring
- Change chunk status
- Set the DATASKIP parameter

You must be logged in as user **root** or user **informix** to execute **onspaces**.

Syntax



Create a Blobspace, Dbspace, Extspace, or Temporary Dbspace



Element	Purpose	Key Considerations
-b <i>blobspace</i>	Names the blobspace to be created.	Restrictions: See restrictions described in “Creating a Blobspace” on page 15-14 . References: Syntax must conform to the Identifier segment; see <i>Informix Guide to SQL: Syntax</i> .
-c	Creates a blobspace, dbspace, or extspace.	References: For background information, see “Creating a DbSPACE” on page 15-9 , “Creating a Blobspace” on page 15-14 , or “Creating a Extspace” on page 15-25 .
-d <i>dbspace</i>	Names the dbspace to be created.	Restrictions: See restrictions described in “Creating a DbSPACE” on page 15-9 . References: Syntax must conform to the Identifier segment; see <i>Informix Guide to SQL: Syntax</i> .
-g <i>pageunit</i>	Specifies the blobspace blobpage size in terms of <i>page_unit</i> , the number of disk pages per blobpage or sbpage.	Restrictions: Unsigned integer. Value must be greater than 0. References: For background information, see “Determining Universal Server Page Size” on page 15-15 .
-l <i>location</i>	Specifies the location of the extspace. The format of this string is determined by the access method.	Restrictions: String. Must not be longer than 255 bytes. References: For background information, see “Creating a Extspace” on page 15-25 .
-m <i>pathname offset</i>	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 <i>pathname</i> and -o <i>offset</i> in this table.	References: For background information, see “Creating a DbSPACE” on page 15-9 and “Creating a Blobspace” on page 15-14 .
-o <i>offset</i>	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 background information, see “Do You Need to Specify an Offset?” on page 15-6 .

(1 of 2)

Element	Purpose	Key Considerations
-p <i>pathname</i>	Indicates the disk partition or device of the initial chunk of the blobspace, or dbspace you are creating.	Additional Information: The chunk must be an existing raw device or cooked 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 Universal Server. References: For pathname syntax, see your operating-system documentation.
-s <i>size</i>	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 specific details on this option, see “Creating a Temporary Dbpace Using the -t Option” in the following section. For background information, see “What Is a Temporary Dbpace?” on page 14-21 .
-x <i>extspace</i>	Names the extspace to be created.	Restrictions: The extspace name must not already exist, and must not exceed eighteen US English characters. References: For background information, see “What Is a Extspace?” on page 14-25 .

(2 of 2)

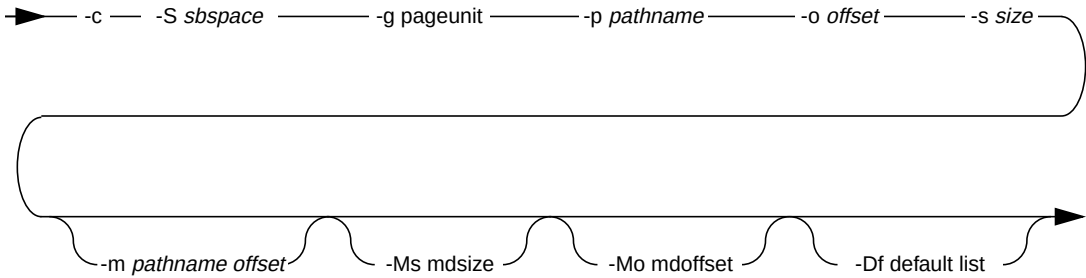
Creating a Temporary Dbpace Using the -t Option

When you create a temporary dbpace with ON-Monitor or **onspaces**, Universal Server does not use the newly created temporary dbpace until you take both of the following steps:

- You add the name of the new temporary dbpace to your list of temporary dbspaces in the DBSPACETEMP configuration parameter, the **DBSPACETEMP** environment variable, or both.
- You reinitialize Universal Server.

Create an Sbspace

Create a Sbspace



Element	Purpose	Key Considerations
-S sbspace	Names the sbspace to be created.	Restrictions: The sbspace name must not already exist, and must not exceed eighteen U.S. English characters. References: Syntax must conform to the Identifier segment; see <i>Informix Guide to SQL: Syntax</i> .
-c	Creates a sbspace.	References: For background information, see “Creating a Sbspace” on page 15-16 .
-g pageunit	Specifies the sbspace sbpage size in terms of <i>page_unit</i> , the number of disk pages per sbpage.	Restrictions: Unsigned integer. Maximum value is 512. References: For background information, see “Determining Universal Server Page Size” on page 15-15 .
-m pathname offset	Specifies an optional pathname and offset to the chunk that will mirror the initial chunk of the new sbspace. Also see the entries for -p pathname and -o offset in this table.	References: For background information, see “Creating a DbSPACE” on page 15-9 and “Creating a Blobspace” on page 15-14 .

(1 of 2)

onspaces: Manage Blobspaces, Dbspaces, Extspaces, or Sbspaces

Element	Purpose	Key Considerations
-Mo <i>mdoffset</i>	Indicates, in kilobytes, the offset into the disk partition or into the device where metadata will be stored.	Restrictions: Integer. Must not be longer than 255 bytes. References: For background information, see “Sizing Sbspace Metadata” on page 15-17.
-Ms <i>mdsize</i>	Specifies the size, in kilobytes, of the metadata area allocated in the initial chunk. The remainder is user data space.	Restrictions: Integer. Must not be longer than 255 bytes. References: For background information, see “Sizing Sbspace Metadata” on page 15-17.
-o <i>offset</i>	Indicates, in kilobytes, the offset into the disk partition or into the device to reach the initial chunk of the sbspace.	Restrictions: Unsigned integer. Value must be greater than 0. References: For background information, see “Do You Need to Specify an Offset?” on page 15-6.
-p <i>pathname</i>	Indicates the disk partition or device of the initial chunk of the sbspace.	Additional Information: The chunk must be an existing raw device or cooked 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 Universal Server. References: For pathname syntax, see your operating-system documentation.
-s <i>size</i>	Indicates, in kilobytes, the size of the initial chunk of the new sbspace.	Restrictions: Unsigned integer. Value must be greater than 0. Size must not exceed 4 Terabytes.
-Df <i>default list</i>	Lists default specifications for smart large objects stored in the sbspace.	Restrictions: Tags are separated by commas. If a tag is not present, system defaults take precedence. The list must be enclosed in double quotation marks (“”) on the command line. References: For a list of tags and their parameters, see Figure 39-2 on page 39-61

(2 of 2)

Creating a Sbspace with the -Df option

When you create an sbspace with the **-Df** option, you can specify several default specifications that affect the behavior of the smart large objects stored in the sbspace. The default specifications must be expressed as a list separated by commas. The list need not contain all of the tags, and **-Df** is optional. The list of tags must be enclosed in double quotation marks (“”). The table in Figure 39-2 describes the tags and their default values.

Figure 39-2
-Df Default Specifications

Tag	Values	System Default	Description
LOGGING	ON or OFF	ON	Specifies the logging mode of smart large objects stored in the sbspace.
ACcesstime	ON or OFF	OFF	When set to ON, Universal Server tracks the time of access to all smart large objects stored in the sbspace.
AVG_LO_SIZE	1 kilobyte to 2 megabytes	64 kilobytes	Specifies the average size, in kilobytes, of the smart large object stored in the sbspace.
EXTENT_SIZE	4 kilobytes to 2 megabytes	16 kilobytes	Specifies the extent size, in kilobytes, of the first smart large object stored in the sbspace.
NEXT_SIZE	4 kilobytes to 2 megabytes	16 kilobytes	Specifies the extent size, in kilobytes, of the next large object stored in the sbspace.
MIN_EXT_SIZE	4 kilobytes to 2 megabytes	4 kilobytes	Specifies the minimum smart large object extent size, in kilobytes.
MAX_IO_SIZE	2 kilobytes to 2 megabytes	2048 bytes	Specifies the maximum size of the I/O block between the client and server when accessing smart large objects stored in the sbspace.

For example, you can create a sbspace with **onspaces** as illustrated in the following example. This example creates a 20-megabyte mirrored sbspace, `eg_sblobsp`, with a sbpage size of 10 kilobytes, where Universal Server page size is 2 kilobytes. An offset of 500 kilobytes for the primary and mirrored chunks is specified, as well as a metadata size of 150 kilobytes with a 200 kilobyte offset is specified. The average expected smart large object size is 32 kilobytes, and the maximum I/O block is 4096 bytes.

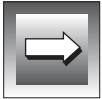
```
% onspaces -c -S eg_sblobsp -g 5 -p /dev/raw_dev1 -o 200 -s 20000 -m /dev/raw_dev2
500 -Ms 150 -Mo 200 -Df "AVG_LO_SIZE=32,MAX_IO_SIZE=4096"
```

The following **-Df** default settings can be overridden in the CREATE TABLE SQL statement:

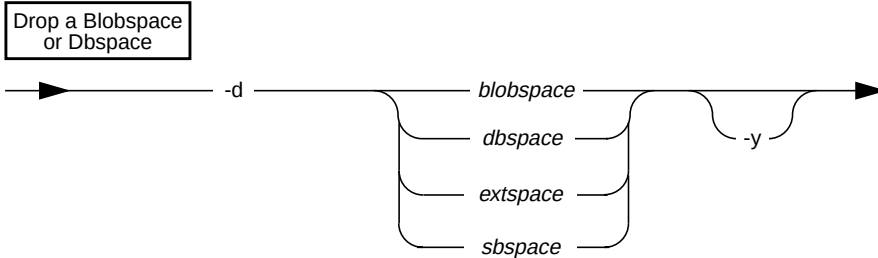
- LOGGING
- ACESSTIME
- EXTENT_SIZE
- NEXT_SIZE

For more information on the CREATE TABLE statement, see the [Informix Guide to SQL: Syntax](#).

Important: When you are adding a chunk to a pre-existing sbspace, the **-Df** option is ignored.



Drop a Blobspace, Dbspace, Extspace, or Sbspace



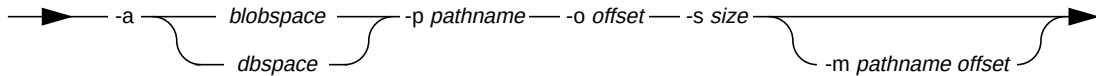
Element	Purpose	Key Considerations
-d	Indicates that either a blobspace, dbspace, extspace, or sbspace is to be dropped.	Restriction: Execute oncheck -pe to verify that no table is currently storing data in the blobspace, dbspace, or sbspace. References: For background information, see “Dropping a Dbpace, Blobpace, or Sbpac” on page 15-19.
-y	Causes Universal Server to automatically respond “yes” to all prompts.	None.
<i>blobspace</i>	Names the blobspace to be dropped.	Additional Information: Before you drop a blobspace, all tables that include a TEXT or BYTE column that references the blobspace must be dropped.
<i>dbspace</i>	Names the dbspace to be dropped.	Additional Information: Before you drop a dbspace, you must first drop all databases and tables that you previously created in the dbspace.
<i>extspace</i>	Names the extspace to be dropped.	Additional Information: A extspace cannot be dropped if it is associated with an existing table or index.
<i>sbspace</i>	Names the sbspace to be dropped.	Additional Information: Before you drop a blobspace, all tables that include a BLOB or CLOB column that references the blobspace must be dropped.



Important: Do not specify a pathname when you are dropping a dbspace, blobpace, extspace, or sbspac.

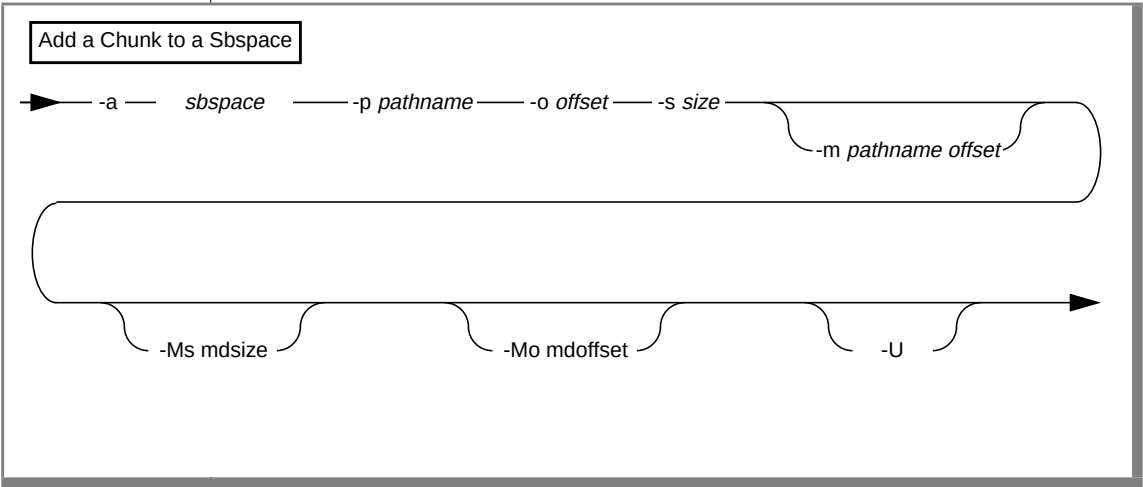
Add a Chunk to a Blobspace or Dbpace

Add a Chunk to a Blobspace or a Dbpace



Element	Purpose	Key Considerations
-a	Indicates that a chunk is to be added.	None.
-m pathname offset	Specifies an optional pathname and offset to the chunk that will mirror the new chunk. Also see the entries for <i>pathname</i> and <i>offset</i> in this table.	References: For background information, see “Adding a Chunk to a Dbpace” on page 15-12 and “Adding a Chunk to a Blobspace” on page 15-16 .
-o offset	After the -a option, <i>offset</i> indicates, in kilobytes, the offset into the disk partition or into the device to reach the initial chunk of the new blobspace or dbpace.	Restrictions: Unsigned integer. Value must be greater than 0. References: Syntax must conform to the Identifier segment; see Informix Guide to SQL: Syntax . For background information, see “Do You Need to Specify an Offset?” on page 15-6 .
-p pathname	Indicates the disk partition or device of the initial chunk of the blobspace or dbpace you are adding.	Additional Information: The chunk must be an existing raw device or cooked 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 Universal 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 dbpace.	Restrictions: Unsigned integer. Value must be greater than 0. Size must not exceed 2 gigabytes.
<i>blobspace</i>	Names the blobspace to which you are adding a chunk.	Restrictions: See restrictions described in “Adding a Chunk to a Blobspace” on page 15-16 . References: Syntax must conform to the Identifier segment; see Informix Guide to SQL: Syntax .
<i>dbpace</i>	Names the dbpace to which you are adding a chunk.	Restrictions: See restrictions described in “Adding a Chunk to a Dbpace” on page 15-12 . References: Syntax must conform to the Identifier segment; see Informix Guide to SQL: Syntax .

Add a Chunk to a Sbpace



Element	Purpose	Key Considerations
-a	Indicates that a chunk is to be added.	None.
-m <i>pathname offset</i>	Specifies an optional pathname and offset to the chunk that will mirror the new chunk. Also see the entries for <i>pathname</i> and <i>offset</i> in this table.	References: For background information, see “Adding a Chunk to a Dbpace” on page 15-12.
-Mo <i>mdooffset</i>	Indicates, in kilobytes, the offset into the disk partition or into the device where metadata should be stored.	Restrictions: String not longer than 255 bytes. References: For background information, see “Sizing Sbpace Metadata” on page 15-17.
-Ms <i>mdspace</i>	Specifies the size, in kilobytes, of the metadata area allocated in the initial chunk. The remainder is user data space.	Restrictions: String not longer than 255 bytes. References: For background information, see “Sizing Sbpace Metadata” on page 15-17.

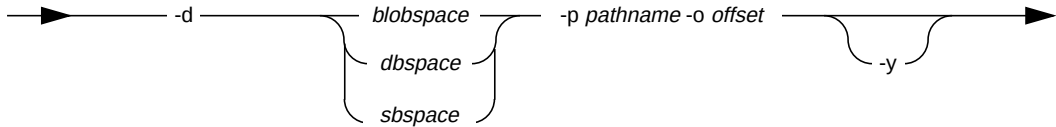
(1 of 2)

Element	Purpose	Key Considerations
-o <i>offset</i>	After the -a option, <i>offset</i> 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 background information, see “ Do You Need to Specify an Offset? ” on page 15-6.
-p <i>pathname</i>	Indicates the disk partition or device of the initial chunk of the sbspace you are creating.	Additional Information: The chunk must be an existing raw device or cooked 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 Universal Server. References: For pathname syntax, see your operating-system documentation.
-U	Specifies that the entire chunk should be used to store user data.	Restrictions: The -M and -U options are mutually exclusive. References: For background information, see “ Adding a Chunk to a Sbspace ” on page 15-18.

(2 of 2)

Drop a Chunk

Drop a Chunk



Element	Purpose	Key Considerations
-d	Drops a chunk.	Restrictions: See restrictions described in “Dropping a Chunk from a Dbpace with onspaces” on page 15-18 or “Dropping a Chunk from a Blobspace or Sbpaces” on page 15-19 .
-o offset	Indicates, in kilobytes, the offset into the disk partition or into the device to reach the initial chunk of the blobspace or dbpace you are dropping.	Restrictions: Unsigned integer. Value must be greater than or equal to 0. References: For background information, see “Do You Need to Specify an Offset?” on page 15-6 .
-p pathname	Indicates the disk partition or device of the initial chunk of the blobspace or dbpace you are dropping.	Additional Information: The chunk must be an existing raw device or cooked 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 Universal Server. References: For pathname syntax, see your operating-system documentation.
-y	Causes Universal Server to automatically respond “yes” to all prompts.	None.

(1 of 2)

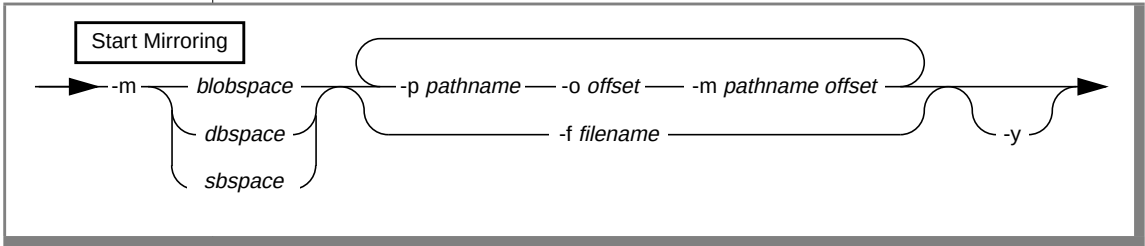
Element	Purpose	Key Considerations
<i>blobspace</i>	Names the blobspace from which the chunk will be dropped.	References: Syntax must conform to the Identifier segment; see <i>Informix Guide to SQL: Syntax</i> . For background information, see “Dropping a Chunk from a Blobspace or Sbspace” on page 15-19.
<i>dbspace</i>	Names the dbspace from which the chunk will be dropped.	References: Syntax must conform to the Identifier segment; see <i>Informix Guide to SQL: Syntax</i> . For background information, see “Dropping a Chunk from a Dbpace with onspaces” on page 15-18.
<i>sbspace</i>	Names the sbspace from which the chunk will be dropped.	References: Syntax must conform to the Identifier segment; see <i>Informix Guide to SQL: Syntax</i> . For background information, see “Dropping a Chunk from a Dbpace with onspaces” on page 15-18.

(2 of 2)



Important: You must specify a pathname to indicate to Universal Server that you are dropping a chunk.

Start Mirroring



Element	Purpose	Key Considerations
-f filename	Indicates to onspaces that chunk-location information is located in a file named <i>filename</i> .	Additional Information: The file must be an existing UNIX cooked file. Pathname must conform to the operating-system-specific rules for pathnames. References: For specific details on this option, see “Using a File to Specify Chunk-Location Information with the -f Option” on page 39-70.
-m	Adds mirroring for an existing dbspace, blobspace, or sbspace.	Additional Information: User data chunks in a mirrored sbspace need not be mirrored.
-m pathname offset	The second time <i>pathname</i> occurs in the syntax diagram, it indicates the disk partition or device of the initial chunk of the blobspace, dbspace, or sbspace that performs the mirroring. The second time <i>offset</i> appears in the syntax diagram, it indicates the offset to reach the mirrored chunk of the newly mirrored dbspace, blobspace, or sbspace. Also see the entries for <i>pathname</i> and <i>offset</i> in this table.	None.
-o offset	The first time <i>offset</i> 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 blobspace, dbspace, or sbspace.	Restrictions: Unsigned integer. Value must be greater than 0. References: For background information, see “Do You Need to Specify an Offset?” on page 15-6.

onspaces: Manage Blobspaces, Dbspaces, Extspaces, or Sbspaces

Element	Purpose	Key Considerations
-p <i>pathname</i>	The first time <i>pathname</i> occurs in the syntax diagram, it indicates the disk partition or device of the initial chunk of the blobspace, dbspace, or sbspace you want to mirror.	Additional Information: The chunk must be an existing raw device or cooked 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 Universal Server. References: For pathname syntax, see your operating-system documentation.
-y	Causes Universal Server to automatically respond “yes” to all prompts.	None.
<i>blobspace</i>	Names the blobspace that you wish to mirror.	References: Syntax must conform to the Identifier segment; see Informix Guide to SQL: Syntax . For background information, see “Starting Mirroring” on page 28-6.
<i>dbspace</i>	Names the dbspace that you wish to mirror.	References: Syntax must conform to the Identifier segment; see Informix Guide to SQL: Syntax . For background information, see “Starting Mirroring” on page 28-6.
<i>sbspace</i>	Names the sbspace that you wish to mirror.	References: Syntax must conform to the Identifier segment; see Informix Guide to SQL: Syntax . For background information, see “Starting Mirroring” on page 28-6.

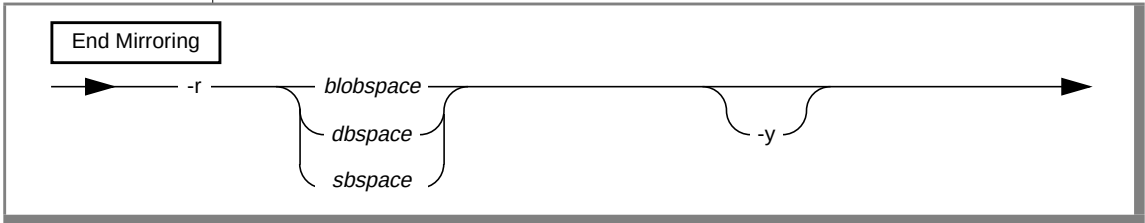
(2 of 2)

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**, you use the **-f** option to indicate to Universal Server that this information is contained in a file whose name you specify in *filename*.

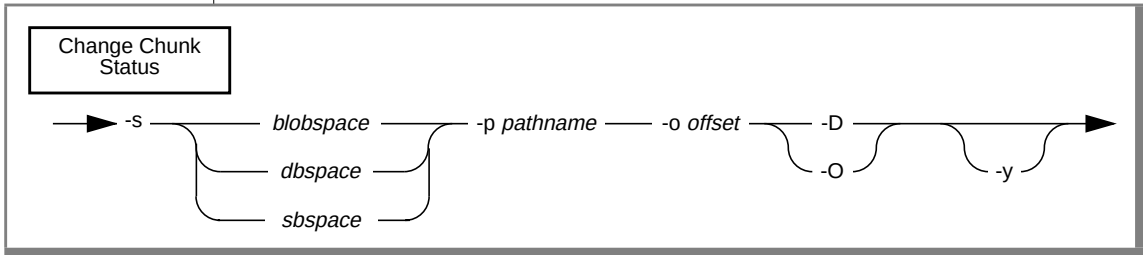
If the dbspace that you are mirroring contains multiple chunks, you must specify a mirrored chunk for each of the primary chunks in the dbspace that you want to mirror. For an example that enables mirroring for a multichunk dbspace, see “Starting Mirroring for Unmirrored Dbspaces with onspaces” on page 28-8.

End Mirroring



Element	Purpose	Key Considerations
-r	Indicates to Universal Server that mirroring should be ended for an existing dbspace, blobspace, or sbpace.	References: For background information, see “Ending Mirroring” on page 28-12.
-y	Causes Universal Server to respond “yes” to all prompts automatically.	None.
<i>blobspace</i>	Names the blobspace for which you wish to end mirroring.	References: Syntax must conform to the Identifier segment; see <i>Informix Guide to SQL: Syntax</i> . For background information, see “Ending Mirroring” on page 28-12.
<i>dbspace</i>	Names the dbspace for which you wish to end mirroring.	References: Syntax must conform to the Identifier segment; see <i>Informix Guide to SQL: Syntax</i> . For background information, see “Ending Mirroring” on page 28-12.
<i>sbpace</i>	Names the sbpace for which you wish to end mirroring.	References: Syntax must conform to the Identifier segment; see <i>Informix Guide to SQL: Syntax</i> . For background information, see “Ending Mirroring” on page 28-12.

Change Status of a Mirrored Chunk



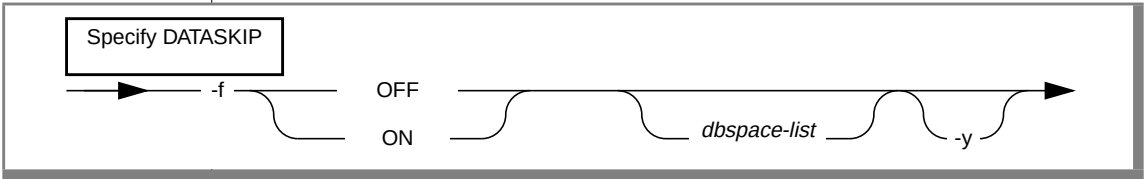
Element	Purpose	Key Considerations
-D	Indicates that you wish to take the chunk down.	None.
-o offset	Indicates, in kilobytes, the offset into the disk partition or into the device to reach the chunk.	Restrictions: Unsigned integer. Value must be greater than or equal to 0. References: For background information, see “Do You Need to Specify an Offset?” on page 15-6.
-O	Indicates that you wish 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 a raw device or a cooked file in a standard UNIX file system. 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 Universal Server. References: For pathname syntax, see your operating-system documentation.
-s	Indicates that you wish to change the status of a chunk.	Restrictions: You can only change the status of a chunk in a mirrored pair. References: For background information, see “Changing the Mirror Status” on page 28-10.
-y	Causes Universal Server to respond “yes” to all prompts automatically.	None.

(1 of 2)

Element	Purpose	Key Considerations
<i>blobspace</i>	Names the blobspace whose status you wish to change.	References: Syntax must conform to the Identifier segment; see <i>Informix Guide to SQL: Syntax</i> . For background information, see “Changing the Mirror Status” on page 28-10 .
<i>dbspace</i>	Names the dbspace whose status you wish to change.	References: Syntax must conform to the Identifier segment; see <i>Informix Guide to SQL: Syntax</i> . For background information, see “Changing the Mirror Status” on page 28-10 .
<i>sbspace</i>	Names the sbspace whose status you wish to change.	References: Syntax must conform to the Identifier segment; see <i>Informix Guide to SQL: Syntax</i> . For background information, see “Changing the Mirror Status” on page 28-10 .

(2 of 2)

Specify DATASKIP Parameter



Element	Purpose	Key Considerations
-f	Indicates to Universal 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 Universal Server message log.
-y	Causes Universal Server to automatically respond “yes” to all prompts.	None.
<i>dbspace-list</i>	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 <i>Informix Guide to SQL: Syntax</i> . For background information, see “Skipping Fragments” on page 17-4 .
OFF	Turns off DATASKIP.	Additional Information: If OFF is used without a <i>dbspace-list</i> , DATASKIP is turned off for all fragments. If OFF is used with a <i>dbspace-list</i> , only the specified fragments are set with DATASKIP off.
ON	Turns on DATASKIP.	Additional Information: If ON is used without <i>dbspace-list</i> , DATASKIP is turned on for all fragments. If ON is used with <i>dbspace-list</i> , only the specified fragments are set with DATASKIP on.

The **onspaces** utility allows the specification of DATASKIP on a dbspace level or across all dbspaces.

onstat: Monitor Universal Server Operation

The **onstat** utility reads shared-memory structures and provides statistics about Universal Server that are accurate at the instant that the command executes. The *system-monitoring interface* also provides information about Universal Server. For information on the system-monitoring interface, see [Chapter 38, “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 39-5.](#)

The following table lists each **onstat** option flag and its function.

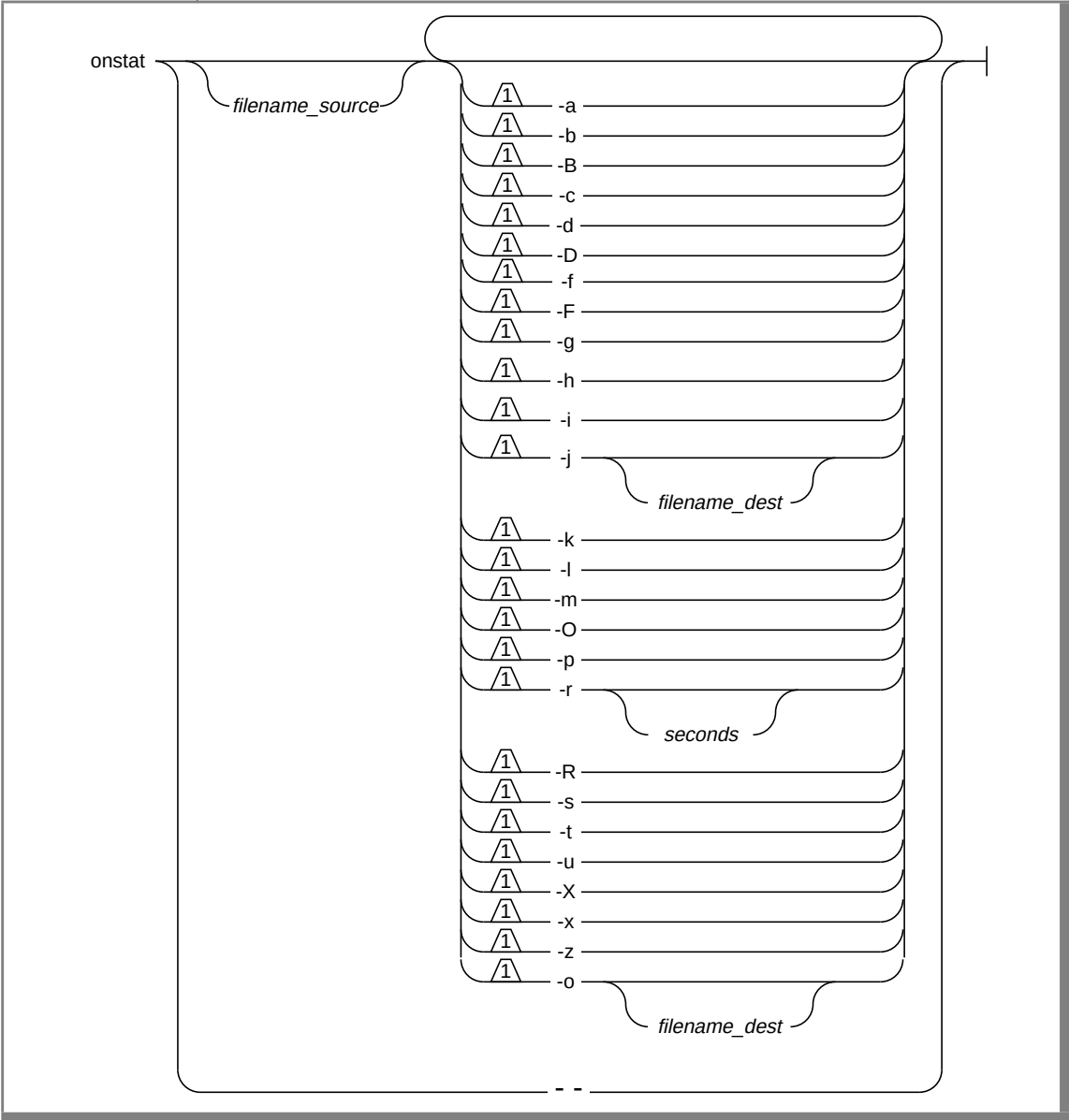
Topic or Function	Option Flag
All onstat options	--
Btree cleaner requests	-C
Buffers, all (in use or not)	-B
Buffers in use	-b
Buffers, includes addresses of waiting threads	-X
Buffer hash chain info	-h
Configuration-file information (\$INFORMIXDIR/etc/\$ONCONFIG)	-c
DATASKIP information	-f
Dbospace chunks, general information	-d
Dbospace chunks, page reads / writes	-D
Interactive mode	-i
Latches	-s
Locks held	-k

(1 of 2)

Topic or Function	Option Flag
Logging information (logical and physical logs, including page addresses)	-l
LRU queues	-R
Monitoring information	-g
Universal Server message log	-m
Universal Server profile of activity	-p
OnLine / Optical 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



Element	Purpose	Key Considerations
--	Displays a listing of all onstat options and their functions. See "-- Option" on page 39-82.	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 39-83.	None.
-b	Displays information about buffers currently in use. See "-b Option" on page 39-83.	None.
-B	Obtains information about all Universal 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: Universal Server first checks if you have assigned a value to the environment variable ONCONFIG . If so, Universal Server displays the contents of \$INFORMIXDIR/etc/\$ONCONFIG . If not, the contents of \$INFORMIXDIR/etc/onconfig are displayed by default.
-C	Prints B-tree cleaner information.	None.
-d	Displays information for chunks in each space. See "-d Option" on page 39-84.	None.
-D	Displays page-read and page-write information for the first 50 chunks in each dbspace. See "-D Option" on page 39-87.	None.
-f	Lists the dbspaces currently affected by the DATASKIP feature. See "-f Option" on page 39-87.	None.
-F	Displays a count for each type of write that flushes pages to disk. See "-F Option" on page 39-87.	None.

(1 of 3)

Element	Purpose	Key Considerations
-g	Monitoring options. See “ -g Monitoring Options ” on page 39-88.	None.
-h	Prints buffer hash chain information.	None.
-i	Puts the onstat utility into interactive mode. See “ -i Option ” on page 39-91.	None.
-k	Displays information about active locks. See “ -k Option ” on page 39-92.	None.
-l	Displays information about physical and logical logs. See “ -l Option ” on page 39-93.	None.
-m	Displays the 20 most-recent lines of the system message log. See “ -m Option ” on page 39-94.	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 <i>filename</i> .	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 INFORMIX-OnLine / Optical memory cache and staging-area blob space. See “ -O Option ” on page 39-95.	None.
-p	Displays profile counts. See “ -p Option ” on page 39-96.	None.
-r	Causes the accompanying onstat options to execute repeatedly after they wait the specified <i>seconds</i> between each execution. The default value of <i>seconds</i> 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 39-99.	None.

Element	Purpose	Key Considerations
-s	Displays general latch information. See “ -s Option ” on page 39-100 .	None.
-t	Displays tblspace information for active tblspaces. See “ -t Option ” on page 39-101 .	None.
-u	Prints a profile of user activity. See “ -u Option ” on page 39-102 .	None.
-x	Displays information about transactions. See “ -x Option ” on page 39-104 .	None.
-X	Obtains precise information about the threads that are sharing and waiting for buffers. See “ -X Option ” on page 39-106 .	None.
-z	Sets the profile counts to zero. See “ -z Option ” on page 39-106 .	None.
<i>filename_dest</i>	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.
<i>filename_source</i>	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” in the following section. For pathname syntax, see your operating-system documentation.
<i>seconds</i>	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 39-91](#).

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)--(Checkpoint)--Up Uptime--Sh_mem Kbytes
```

<i>Version</i>	is the product version number.				
<i>Mode</i>	is the current operating mode.				
<i>(Type)</i>	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.				
<i>(Checkpoint)</i>	is a checkpoint flag. If it is set, the header might display two other fields after the mode if the timing is appropriate: <table data-bbox="504 1047 1240 1242"> <tr> <td>(CKPT REQ)</td><td>indicates that some Universal Server user thread has requested a checkpoint.</td></tr> <tr> <td>(CKPT INP)</td><td>indicates that a checkpoint is in progress. During the checkpoint, access is limited to read only. Universal Server cannot write or update data until the checkpoint ends.</td></tr> </table>	(CKPT REQ)	indicates that some Universal 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. Universal Server cannot write or update data until the checkpoint ends.
(CKPT REQ)	indicates that some Universal 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. Universal Server cannot write or update data until the checkpoint ends.				
<i>Uptime</i>	indicates how long the database server has been running.				
<i>Sh_mem</i>	is the size of Universal Server shared memory, expressed in kilobytes.				

An example header follows:

```
INFORMIX-Universal Server Version 9.10.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:

```
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	Universal 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. (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: <table> <tr> <td>0x01</td><td>Modified data</td></tr> <tr> <td>0x02</td><td>Data</td></tr> <tr> <td>0x04</td><td>LRU</td></tr> <tr> <td>0x08</td><td>Error</td></tr> </table>	0x01	Modified data	0x02	Data	0x04	LRU	0x08	Error										
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: <table> <tr> <td>1</td><td>data page</td></tr> <tr> <td>2</td><td>tblspace page</td></tr> <tr> <td>4</td><td>free-list page</td></tr> <tr> <td>8</td><td>chunk free-list page</td></tr> <tr> <td>9</td><td>remainder data page</td></tr> <tr> <td>b</td><td>partition resident blobpage</td></tr> <tr> <td>c</td><td>blobspace resident blobpage</td></tr> <tr> <td>d</td><td>simple-large-object chunk free-list bit page</td></tr> <tr> <td>e</td><td>simple-large-object chunk blob map page</td></tr> </table>	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	simple-large-object chunk free-list bit page	e	simple-large-object chunk blob map page
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	simple-large-object chunk free-list bit page																		
e	simple-large-object 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 39-106 .	

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.

-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
		S -- Sbspace
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 Universal Server instance including the rootdbs) and the number of total spaces.

Active spaces refers to the current number of spaces in the Universal Server instance including the rootdbs. Total refers to total *allowable* spaces for this Universal 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.) For a sbspace indicates free pages of userdata space and total userdata space.
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 S -- Sbspace 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 Universal 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 simple large objects, **onstat -d** shows that the pages where the obsolete simple large objects 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 a simple large object from a blob space, **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 data-skip feature. The **-f** option lists both the dbspaces that were set with the DATASKIP configuration parameter and the **-f** option of **onspace**. When you execute **onstat -f**, Universal Server displays one of the following three outputs:

- Data skip is OFF for all dbspaces.
- Data skip is ON for all dbspaces.
- Data skip 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 a foreground write occurred.
LRU Writes is the number of times an LRU write occurred.
Chunk Writes is the number of times 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 Universal 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. Universal 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 <code>state</code> field. If <code>state</code> is C, <code>data</code> is the chunk number to which the page cleaner is writing buffers. If <code>state</code> is L, <code>data</code> is the LRU queue from which the page cleaner is writing. The <code>data</code> 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 <i>pool name</i> <i>session id</i>	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 <i>rstcb</i> value corresponds to the <i>user</i> field of the onstat -u command.
con	Prints conditions with waiters.

dic <i>table</i>	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 Data-Replication Status” on page 33-76 for a description of the information that is displayed.)
dsc	Prints data-distribution cache information.
ffr <i>pool name</i> <i>session id</i>	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 <i>pool name</i> <i>session id</i>	Prints memory statistics for a pool(s). 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 <i>client id</i>	Prints shared-memory status by <i>client id</i> . If <i>client id</i> 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 <i>session id</i>	Prints network shared-memory status by <i>session id</i> . If <i>session id</i> 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 \$INFORMIXDIR/etc/.infos.DBSERVERNAME file.
ppf <i>partition number</i> 0	Prints partition profile for <i>partition number</i> ; 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 <i>session id</i>	Prints session information by <i>session id</i> . 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 <i>no-age</i> or <i>processor</i> affinity features.
sql <i>session id</i>	Prints SQL information by <i>session id</i> . If <i>session id</i> is omitted, a one-line summary for each session is printed.
stk <i>tid</i>	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 <i>tid</i>	Prints thread profile for <i>tid</i> ; 0 prints profiles for all threads.

ufr <i>pool name</i> <i>session id</i>	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-Universal Server Version 9.10.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 <code>wait</code> 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 <code>address</code> 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 INFORMIX-OnLine/Optical 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. Universal 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 Universal Server allocated to the cache.
avail	describes how much of alloc (in kilobytes) is not used.
number	is the number of simple large objects that Universal Server successfully put into the cache without overflowing.
kbytes	is the number of kilobytes of the simple large objects that Universal Server put into the cache without overflowing.
number	is the number of simple large objects that Universal Server wrote to the staging-area blobspace.
kbytes	is the number of kilobytes of the simple large objects that Universal Server wrote to the staging-area blobspace.

Although the **size** output indicates the amount of memory that is specified in the configuration parameter OPCACHEMAX, Universal Server does not allocate memory to OPCACHEMAX until necessary. Therefore, the **alloc** output reflects only the number of 1-kilobyte pieces of the largest simple large object that has been processed. When the values in the **alloc** and **avail** output are equal 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, Universal Server uses the size that you specify in the configuration parameter OPCACHEMAX .
number	is the number of simple large objects that Universal Server put into cache without overflowing.
kbytes	is the number of kilobytes of the simple large objects that Universal Server put into the cache without overflowing.
number	is the number of simple large objects that Universal Server wrote to the staging-area blobspace.
kbytes	is the number of kilobytes of the simple large objects that Universal 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 Universal 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.

Universal 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 $100 * (\text{bufreads} - \text{dskreads}) / \text{bufreads}$. (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 $100 * (\text{bufwrits} - \text{dskwrits}) / \text{dskwrits}$. (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 Universal 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 a resource was requested when none was available:

ovlock	is the number of times that Universal 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 a user thread must wait for a buffer.
lokwaits	increments each time a user thread must wait for a lock.
lockreqs	increments each time a lock is requested.
deadlks	increments each time a potential deadlock is detected and prevented.
dltouts	increments each time 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 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 Universal 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.

-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 12-36](#).

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 12-36 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, Universal Server attempts to use buffers from the FLRU queue rather than MLRU queue. (A modified buffer must be written to disk before Universal Server can use the buffer.) - F free LRU with fewest elements. Universal 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	what 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 <code>wait</code> 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 machine 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. 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: <table> <tr> <td>0x01</td><td>Busy</td></tr> <tr> <td>0x02</td><td>Dirty (modified pages that have not been flushed to disk)</td></tr> </table>	0x01	Busy	0x02	Dirty (modified pages that have not been flushed to disk)
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 a next extent has been allocated.

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: <table><tr><td>B</td><td>waiting on a buffer</td></tr><tr><td>C</td><td>waiting on a checkpoint</td></tr><tr><td>G</td><td>waiting on a write of the logical-log buffer</td></tr><tr><td>L</td><td>waiting on a lock</td></tr><tr><td>S</td><td>waiting on mutex</td></tr><tr><td>T</td><td>waiting on a transaction</td></tr><tr><td>Y</td><td>waiting on condition</td></tr><tr><td>X</td><td>waiting on a transaction cleanup (rollback)</td></tr></table> The flag codes for position 2: <table><tr><td>*</td><td>transaction active during an I/O failure</td></tr></table> The flag codes for position 3: <table><tr><td>A</td><td>dbspace backup thread</td></tr></table>	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)	*	transaction active during an I/O failure	A	dbspace backup thread
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)																				
*	transaction active during an I/O failure																				
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 Transaction is XA-prepared. Universal Server (in an X/Open DTP environment) is prepared to commit or is currently in the process of committing.
	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 39-3 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 Universal Server. For example, the last line of a sample **onstat -u** output is shown:

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 Universal Server was 17.

INFORMIX-Universal Server Version 9.10.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 39-3
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 Universal 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 39-4 shows output from **onstat -x**. The last line of the **onstat -x** output displays the maximum number of concurrent transactions since you initialized Universal Server. For example, the last line of a **onstat -u** output is shown:

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 Universal Server was 6.

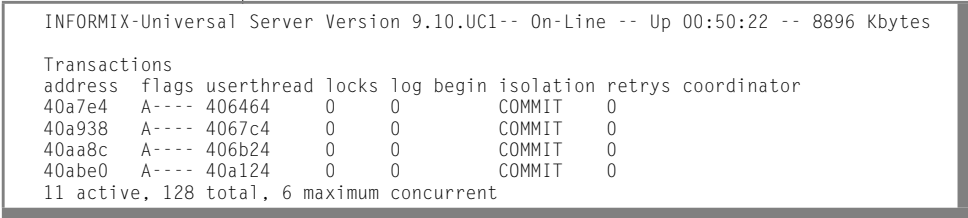


Figure 39-4
Output from
onstat -x

-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 Universal Server manages. Any user can reset the profile counts and thus interfere with monitoring that another user is conducting.

ontape: Logging, Archives, and Restore

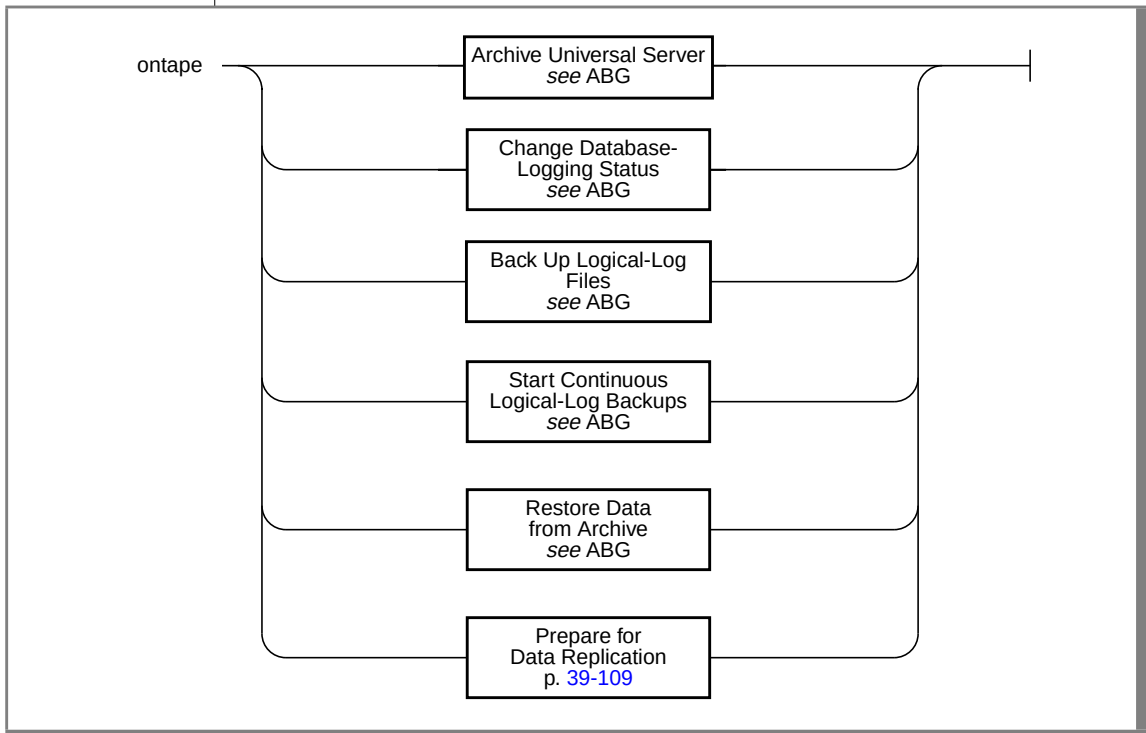
The **ontape** utility allows you to perform several different tasks. You also can perform all the tasks that the **ontape** utility performs by using either the ON-Bar backup and restore system or ON-Archive. For more information on ON-Bar, refer to the [INFORMIX-Universal Server Backup and Restore Guide](#). For more information on the ON-Archive utility, refer to the [INFORMIX-Universal Server Archive and Backup Guide](#).

You can use the **ontape** utility to perform the following tasks:

- Archive data that Universal 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 the [INFORMIX-Universal Server Archive and Backup Guide](#). Syntax for **ontape** options other than **-t** and **-l** appear in Chapter 13 of that manual.

Things to Consider

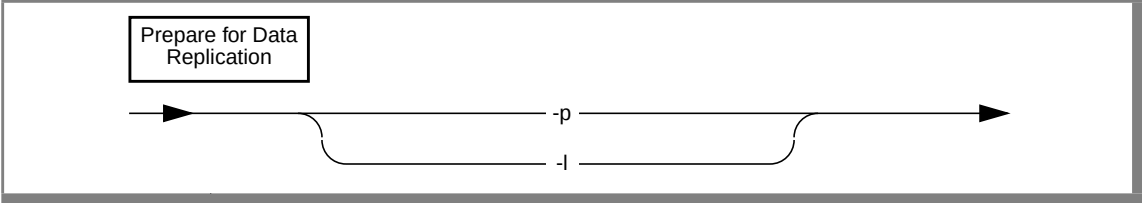
If more than one tape is needed during data replication, **ontape** prompts for each additional tape. Do not run **ontape** in background mode (using the UNIX **&** operator) because you might need to provide input from the terminal or window.

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 Universal Server 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 Universal Server database server.	Additional Information: This option is used expressly for replicating data prior to initiating the data-replication feature.

The **-p** and **-l** options are used for replicating data initially in a pair of database servers that use data replication. See [“Starting Data Replication for the First Time” on page 30-10](#).

Message-Log Messages

How the Messages Are Ordered in This Chapter	40-3
Message Categories	40-4
Messages: A-B	40-4
Messages: C	40-6
Messages: D-E-F	40-14
Messages: G-H-I	40-18
Messages: J-K-L-M	40-20
Messages: N-O-P	40-26
Messages: Q-R-S	40-33
Messages: T-U-V	40-38
Messages: W-X-Y-Z	40-41
Messages: Symbols	40-42

T

his chapter contains nonnumbered messages that are printed in the INFORMIX-Universal 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, you can look up their explanations and corrective actions in the *Informix Error Messages* manual.

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 Universal 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 33-6.

For error messages that pertain to ON-Bar, refer to the *INFORMIX-Universal Server Backup and Restore Guide*. For error messages that pertain to ON-Archive, refer to the *INFORMIX-Universal Server Archive and Backup Guide*.

How the Messages Are Ordered in This Chapter

Universal 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 40-43.

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 Universal Server entity
Action: What action the Universal Server administrator should take
See Also: DUMPDIR/af.uniqid containing more diagnostics.

Cause: 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 technical support.

Begin recreating indexes deferred during recovery.

Cause: During recovery, indexes to be created are deferred until after recovery completes. This message indicates that Universal 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 the ONCONFIG file). 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 the ONCONFIG file) 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 Universal Server fails in between the two, the fast recovery process will attempt to complete the commit the next time that you initialize Universal Server.
- If this completion of the commit fails, Universal Server generates the preceding message.
- Action:** Examine the logical log as described in [Chapter 41, “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 non-zero, 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 tbspace tbspace 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. Since 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*, *errno* = *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:** Refer to your operating-system documentation for more information about corrective actions.

Cannot Open Primary Chunk *pathname*, *errno* = *nnn*

- Cause:** The primary chunk of a mirrored pair cannot be opened. The chunk *pathname* and the operating-system error are returned.
- Action:** Refer to 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:** Refer to your operating-system documentation.

shutdown

- Cause:** A dbspace has gone down during a checkpoint interval. The Universal Server is configured to wait for an override when this occurs.
- Action:** Either shutdown Universal Server or issue an **onmode -O** command to override the down dbspace. For more information on the **onmode** utility, refer to [“onmode: Mode and Shared-Memory Changes” on page 39-35](#)

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 0). 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: Refer to 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: Refer to 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/BL0Bspace 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:** Status message informing 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:** Refer to 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.

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.

INFORMIX-Universal 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-Universal Server Initialized - Shared Memory Initialized

- Cause:** Shared memory has been initialized.
- Action:** None required.

INFORMIX-Universal 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. Refer to 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: Refer to the error message 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* logging subsystem (OLDRSAM, RSAM, SBLOB, RTREE)
 - ???? 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* logging subsystem (OLDRSAM, RSAM, SBLOB, RTREE)
 - snum* subsystem number

Cause: *xx* is the transaction number that appears in the logical log.

Action: The database server detected an error during the rollforward portion of fast recovery or logical-log restore.

Action: Notify Informix Technical Support.

Log record (*type:subtype*) at log *nn*, *0xn* was not undone

The log record that caused the error is identified as follows:

type is the logical-log record type.

subtype logging subsystem (OLDRSAM, RSAM, SBLOB, RTREE)

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*

The log record that caused the error is identified as follows:

type is the logical-log record type.

subtype logging subsystem (OLDRSAM, RSAM, SBLOB, RTREE)

pnum part number

rid row id

num iserror number

Cause: Logging failure

Action: Contact Informix Technical Support.

Log record (*type:subtype*) in log *nn*, offset *0xn* was not rolled back

The log record that caused the error is identified as follows:

<i>type</i>	is the logical-log record type.
<i>subtype</i>	logging subsystem (OLDRSAM, RSAM, SBLOB, RTREE)
<i>log</i>	is the logical-log log ID where the record is stored.
<i>offset</i>	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:** Refer to 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 35, “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 for assistance.

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 for assistance. To correct item 3, increase the SHMTOTAL value in your ONCONFIG configuration file. Refer to 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: Universal Server detected an error in an attempt to acquire more memory space from the operating system.

Action: Refer to 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 the ONCONFIG file.

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 Universal 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 non-zero 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 non-zero 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 Universal 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 $m \leq n$, $m \geq 0$, and $n \geq 0$

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- 10000.

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 Universal 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 Universal Server cannot detect the existence of the optical-storage subsystem.
- Action:** Check that the optical subsystem is on-line.

Optical Subsystem STARTUP Error.

- Cause:** Universal Server detects that the optical-storage subsystem is running, but Universal 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 Universal Server instance.
- Action:** You are unlikely to receive this message. If the error persists after you restart the database server, call Informix Technical Support.

Open transaction detected when changing log versions.

- Cause:** Universal Server detected an open transaction while it was trying to convert the data from a previous version of Universal 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:** Refer to the log file for additional information.

out of resident shared memory

- Cause:** Could not allocate more memory for the specified segment.
- Action:** Refer to the log file for additional information.

out of virtual shared memory

- Cause:** Could not allocate more memory for the specified segment.
- Action:** Refer to the log file for additional information.

PANIC: Attempting to bring system down

- Cause:** A fatal database server error occurred.
- Action:** Refer to 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 Universal Server database servers where the result of a transaction is unknown.
- Action:** See [“Determine If a Transaction Was Implemented Inconsistently” on page 35-4.](#)

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:** Universal Server is attempting to contact remote site *server_name*. For some unknown reason, Universal 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: Universal Server has entered quiescent mode from some other state. Only users logged in as **informix** or as **root** can interact with Universal Server. No user can access a database.

Action: None required.

Recovery Mode

Cause: Universal 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".tablename-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, Universal Server cannot roll forward a specific logical-log record. Universal Server might be able to change to quiescent or on-line mode, but some inconsistency could result. Refer to 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 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 Universal Server is attempting to commit a transaction that has no current owner, and the transaction develops into a long transaction. Universal 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 Universal Server is attempting to commit a distributed transaction that has no current owner, and the transaction develops into a long transaction. Universal 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: Refer to your operating-system documentation.

`semget: errno = nn`

Cause: An allocation of a semaphore set failed. The UNIX operating-system error is returned.

Action: Refer to 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 Universal Server tried to remove or lock a shared-memory segment. The UNIX operating-system error number is returned.

Action: Refer to your operating-system documentation.

`shmdt: errno = nn`

Cause: An error occurred while Universal Server was trying to detach from a shared-memory segment. The UNIX operating-system error number is returned.

Action: Refer to your operating-system documentation.

`shmem sent to filename`

Cause: Universal 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:** Universal 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:
- spacenam* is the name of the dbsapce, blobspace, or sbSPACE that Universal Server is recovering.
- addr* is the address of the RSAM control block.
- Cause:** Informational message. Indicates that Universal 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:
- spacenam* is the name of the dbsapce, blobspace, or sbSPACE that Universal Server has recovered.
- addr* is the address of the RSAM control block.
- Cause:** Informational message. Indicates that Universal Server recovered the space.
- Action:** None required.

Space *spacename* -- Recovery Failed(*addr*)

Cause: Informational message. Indicates that Universal Server was unable to recover the space.

Action: None required.

Description: The variables in this message have the following values:

spacenam is the name of the dbsapce, blobspace, or sbSPACE that Universal Server failed to recover.

addr is the address of the RSAM control block.

Cause: Informational message. Indicates that Universal Server was unable to recover the space.

Action: None required.

sysmaster database built successfully

Cause: Universal Server successfully built the **sysmaster** database.

Action: None required.

Messages: T-U-V

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 Universal Server has been attempted while an in-place alter table is in progress. The previous versions of Universal 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 Universal 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:** Refer to 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* 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 Universal 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: Universal Server encountered an out of memory condition.

Action: No action is necessary.

Unable to open tblspace *nn*, iserrno = *nn*

Cause: Universal Server cannot open the specified tblspace. (The value *nn* is the hexadecimal representation of the tblspace number.)

Action: Refer to the ISAM error message number *nn*, which should explain why the tblspace cannot be accessed. The error message is contained in the [Informix Error Messages](#) manual.

Virtual processor limit exceeded

Cause: You configured Universal Server with more than the maximum number of virtual processors allowed (1000).

Action: To reduce the number of virtual processors, decrease the values set by VPCLASS, NUMCPUVPS, NUMAIOVPS, or NETTYPE in your ONCONFIG configuration 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 may 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 Universal 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 Universal 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. Universal 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, refer to [“BUFFERS” on page 37-9](#).

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.

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).

Interpreting Logical-Log Records

Reading Logical-Log Records	41-3
Transactions That Drop a Table or Index	41-4
Transactions That Are Rolled Back	41-4
Checkpoints with Active Transactions	41-5
Distributed Transactions	41-5
Logical-Log Record Structure	41-6
Logical-Log Record Header	41-6
Logical-Log Record Types and Additional Columns	41-7

The **onlog** utility (see [“onlog: Display Logical-Log Contents” on page 39-30](#)) and the ON-Archive command LIST/LOGRECORDS (described in the *INFORMIX-Universal Server Archive and Backup Guide*) display the logical-log records that the logical-log files contain. 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 INFORMIX-Universal 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 39, “Utilities.”](#)

Reading Logical-Log Records

Most SQL statements generate multiple logical-log records. Interpreting logical-log records is more complicated when Universal 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 Universal 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. Universal Server completes all the other parts of the transaction and writes the relevant logical-log records.
2. Universal Server writes a BEGCOM record to the logical log and the records associated with the DROP TABLE or DROP INDEX (DINDEX, for example).
3. Universal Server writes a COMMIT record.

If the transaction is terminated unexpectedly after the database server writes the BEGCOM record to the logical log, Universal 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. Universal 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

These logical-log records are described further under [“Two-Phase Commit and Logical-Log Records” on page 34-30](#).

If you are performing distributed transactions with INFORMIX-TP / XA, the PREPARE record is replaced by the XAPREPARE 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 41-7](#).

Logical-Log Record Header

Figure 41-1 contains sample output that illustrates the header columns that display for a logical-log record.

Figure 41-1
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 41-2 defines the contents of each header column.

Figure 41-2
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 41-3 on page 41-8](#) 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 41-6](#).

Figure 41-3
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.		
BEGIN	Begin work.	date	decimal
		time	decimal
		PID	decimal
		user	ASCII

(1 of 14)

Logical-Log Record Types and Additional Columns

Record Type	Action	Additional Columns	Format
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
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

(2 of 14)

Logical-Log Record Types and Additional Columns

Record Type	Action	Additional Columns	Format
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
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
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

(3 of 14)

Record Type	Action	Additional Columns	Format
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
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	

Logical-Log Record Types and Additional Columns

Record Type	Action	Additional Columns	Format
CKPOINT	Checkpoint.	max users	decimal
		number of active transactions	decimal
CLR	Compensation-log record; part of a rollback.	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, number of extents in lomaphdr	hexadecimal decimal
DELETE	Delete before-image.	tblspace ID	hexadecimal
		rowid	hexadecimal
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

(5 of 14)

Record Type	Action	Additional Columns	Format
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
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	
ERASE	Drop tblspace.	tblspace ID	hexadecimal

(6 of 14)

Logical-Log Record Types and Additional Columns

Record Type	Action	Additional Columns	Format
EXTEND	Extent added to an extent list of a smart large object	smart-large-object ID	hexadecimal
		extent	decimal
		lomap overflow page number	decimal
HDELETE	Home row delete.	tblspace ID	hexadecimal
		rowid	hexadecimal
		slotlen	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)
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

(7 of 14)

Logical-Log Record Types and Additional Columns

Record Type	Action	Additional Columns	Format
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
IDXFLAGS	Index flags.	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

(8 of 14)

Logical-Log Record Types and Additional Columns

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 blob page delete.	bpageno	hexadecimal
		status	USED / FREE
		unique ID	decimal
PBINSERT	Tblspace blob page insert.	bpageno	hexadecimal
		tblspace ID	hexadecimal
		rowid	hexadecimal
		slotlen	decimal
		pbrowid	hexadecimal
PDELETE	Queue a smart large object for deletion at commit	smart large object ID	hexadecimal
PDINDEX	Predrop index.	tblspace ID	hexadecimal
PGALTER	Page altered in-place.	tblspace ID	hexadecimal
		physical page number	hexadecimal

(9 of 14)

Logical-Log Record Types and Additional Columns

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

(10 of 14)

Logical-Log Record Types and Additional Columns

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
PTRUNC	Queue a smart large object for truncation at commit	smart large object ID	hexadecimal
RDELETE	Remainder page delete.	tblspace ID	hexadecimal
		rowid	hexadecimal
		slotlen	decimal
REFCOUNT	increment or decrement the reference count of a smart large object.	smart-large-object ID	hexadecimal
		1 if increment; 0 if decrement	decimal

(11 of 14)

Logical-Log Record Types and Additional Columns

Record Type	Action	Additional Columns	Format
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
ROLLBACK	Rollback work.	date	decimal
		time	decimal
RSVEXTEN	Logs the extension to the reserved pages.	no. of pages	decimal
		physical page number of extent	hexadecimal
RUPAFT	Remainder page update, after-image.	tblspace ID	hexadecimal
		rowid	hexadecimal
		slotlen	decimal
RUPBEF	Remainder page update, before-image.	tblspace ID	hexadecimal
		rowid	hexadecimal
		slotlen	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	hexadecimal
		rowid	hexadecimal
		forward ptr rowid	hexadecimal
		old slotlen	decimal
		new slotlen	decimal
		no. of pieces	decimal

(12 of 14)

Record Type	Action	Additional Columns	Format
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.		
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 tablespaces (by tblspace number) held by the transaction. (In a distributed transaction, transactions are shown as the owners of locks.)	no. of locks	decimal
		tblspace number	hexadecimal
UDINSERT	Append of new user data	chunk	decimal
		page within chunk	hexadecimal
		offset within page	hexadecimal
		data length	hexadecimal
UDUPAFT	Update of user data after image if a UDWRITE is too expensive.	chunk	decimal
		page within chunk	hexadecimal
		offset within page	hexadecimal
		data length	hexadecimal
UDUPBEF	Update of user data before image if a UDWRITE is too expensive.	chunk	decimal
		page within chunk	hexadecimal
		offset within page	hexadecimal
		data length	hexadecimal

(13 of 14)

Record Type	Action	Additional Columns	Format
UDWRITE	Update of user data (difference image)	chunk	decimal
		page within chunk	hexadecimal
		offset within chunk	hexadecimal
		length before write	hexadecimal
		length after write	hexadecimal
UNDO	Header record to a series of transactions to be rolled back.	count	decimal
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	hexadecimal
		unique ID	decimal
UPDAFT	Update after-image.	tblspace ID	hexadecimal
		rowid	hexadecimal
UPDBEF	Update before-image.	tblspace ID	hexadecimal
		rowid	hexadecimal
XAPREPARE	Participant can commit this XA transaction.	none	

(14 of 14)

Disk Structures and Storage

Dbspace Structure and Storage	42-4
Structure of the Root Dbspace	42-4
Reserved Pages	42-6
Structure of a Regular Dbspace	42-15
Structure of an Additional Dbspace Chunk	42-16
Structure of a Mirrored Chunk	42-16
Structure of the Chunk Free-List Page	42-17
Chunk Free-List Page Entries	42-18
Creation of Free-List Pages	42-18
Structure of the Tblspace Tblspace	42-19
Tblspace Tblspace Entries	42-19
What Is the Tblspace Number?	42-20
Tblspace Number Elements	42-21
Tblspace Tblspace Size	42-22
Tblspace Tblspace Bit-Map Page	42-22
Structure of the Database Tblspace	42-23
What Is the Database Tblspace Number?	42-23
Database Tblspace Entries	42-23
Structure of a Dbspace Bit-Map Page	42-24
Types of Bit-Map Entries	42-24
Two-Bit Bit-Map Pages	42-25
Four-Bit Bit-Map Pages	42-26
Structure and Allocation of an Extent	42-27
Extent Structure	42-27
Next-Extent Allocation	42-31
Structure and Storage of a Dbspace Page	42-34
Structure of a Dbspace Page	42-34
Rowid in Nonfragmented Tables	42-37
Rowid in Fragmented Tables	42-39
Informix Recommendations on Use of Rowid	42-40
Data-Row Format and Storage	42-40

Structure of B-Tree Index Pages	42-48
Definition of Terms in B-Tree Indexing	42-48
Logical Storage of Indexes	42-50
Physical Storage of Indexes	42-55
Physical Storage Format of Index Pages	42-56
Functional Indexes	42-60
Structure of R-Tree Index Pages	42-60
Blobspace Structure and Storage	42-62
Structure of a Blobspace	42-62
Simple-Large-Object Storage and the Simple-Large-Object	
Descriptor	42-64
When Are Simple Large Objects Created?	42-64
Are Simple Large Objects Modified?	42-65
What Limits the Simple-Large-Object Size?	42-65
Structure of a Dbospace Blobpage	42-65
Blobspace Page Types.	42-67
What Is the Blobspace Free-Map Page?	42-67
What Is the Blobspace Bit-Map Page?	42-67
What Is the Blobpage?	42-68
Structure of a Blobspace Blobpage	42-68
Blobpage Structure	42-68
What Is in the Blobpage Header?	42-69
Sbospace Structure	42-71
Metadata Area Structure.	42-73
Sbpage Structure	42-74
A Multiple Chunk Sbospace	42-75
Database and Table Creation: What Happens on Disk	42-76
Creating a Database	42-76
Disk-Space Allocation for System Catalog Tables	42-76
System Catalog Tables Are Tracked.	42-76
Creating a Table.	42-77
Disk-Space Allocation	42-77
Entry Is Added to Tblspace Tblspace	42-78
Entries Are Added to the System Catalog Tables	42-78
What Happens on Disk When a Temporary Table Is	
Created?	42-80

T

he INFORMIX-Universal Server achieves its high performance by managing its own I/O. Universal Server manages storage, search, and retrieval. As Universal Server stores data, it creates the structures it needs to search for and retrieve the data later. Universal Server disk structures also store and track control information needed to manage logging and backups. Universal 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 14, “Where Is Data Stored?”](#)

This chapter discusses the following topics related to Universal Server disk data structures:

- Simple-large-object storage and simple-large-object 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 dbospace structure
- Rowid structure
- Sbospace structure
- Sbospace chunk header structure
- Sbospace metadata area structure
- Sbospace sbpage structure
- Multiple-chunk sbospace 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

Dbospace Structure and Storage

This section explores the disk structures and storage techniques that Universal Server uses to store data in a dbospace.

Structure of the Root Dbospace

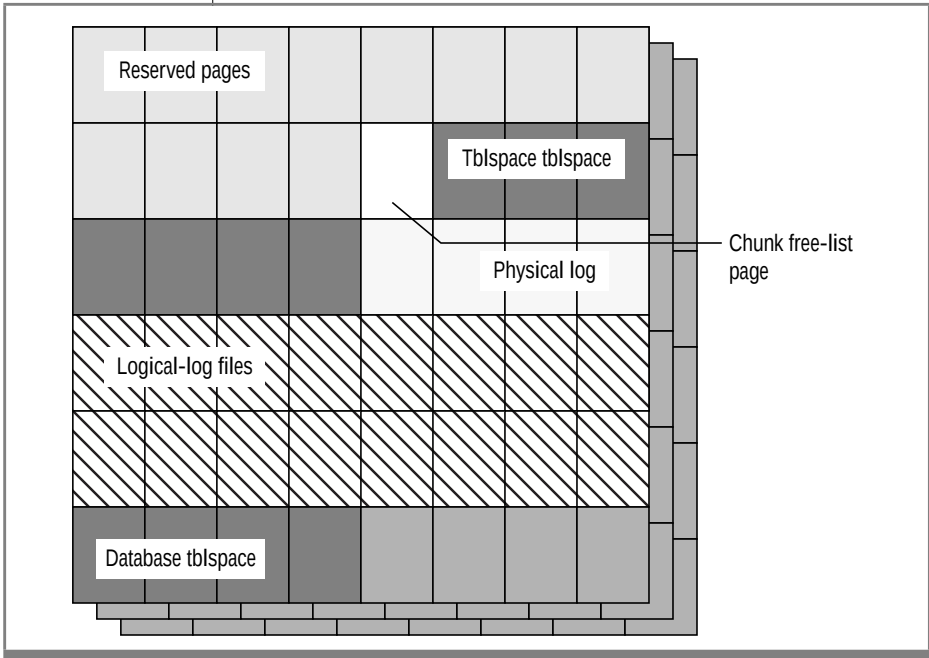
The Universal Server configuration file contains the location of the initial chunk of the root dbospace. If the root dbospace is mirrored, the mirror-chunk location is also specified in the configuration file.

As part of disk-space initialization, Universal Server initializes the following structures in the initial chunk of the root dbospace:

- Twelve reserved pages
- The first chunk free-list page
- The tblspace tblspace
- The database tblspace
- The physical log
- The logical-log files

Figure 42-1 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 42-1
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 Universal Server. Figure 42-2 provides the name and a brief description of each of the reserved pages.

Figure 42-2
Reserved Pages and Their Use

Order	Page Name	Page Usage
1	PAGE_PZERO	System identification
2	PAGE_CONFIG	Copy of configuration 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 Universal Server begins to update the reserved pages as part of the checkpoint procedure.

Universal Server writes to PAGE_PZERO on the following three occasions:

- At boot time
- When you convert to Universal Server Version 9.1 from Version 7.2x of OnLine
- When you convert from OnLine Version 7.2x to a previous version of OnLine

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, Universal Server writes the latest checkpoint information to one of the pages in the pair. During the next checkpoint, Universal 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.

Universal Server follows a different procedure for updating information in the next three reserved-page pairs. Universal Server updates the dbspace, primary chunk, or mirror-chunk reserved pages only when a change occurs. Universal 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, Universal Server checks each shared-memory descriptor table for a change flag.

If the flag is set, Universal Server prepares to write the modified descriptor information to the appropriate page in the reserved-page pair. First, Universal Server switches from the current page (which is the page that received the last write) to the other page in the pair. Second, Universal Server writes the information to the reserved page. Third, Universal 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. Universal 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	Universal 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 either a copy of the ONCONFIG configuration file specified by **\$INFORMIXDIR/etc/\$ONCONFIG** or, if ONCONFIG is not set, the file **\$INFORMIXDIR/etc/onconfig**, by default.

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.

Universal Server uses the checkpoint and logical-log file information for checkpoint processing. The date and time of the last checkpoint, available from the Force-Ckpt menu of ON-Monitor, is obtained from this reserved page.

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 Universal Server 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.

Universal 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.

Universal 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 (blobospace 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 blobospace.
0x400	Chunk is being dropped.
0x800	Chunk is part of staging blobospace.
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.

Universal 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 (blobospace 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 blobospace.

(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.

Universal Server uses the PAGE_ARCH reserved pages to describe the most-recent and the second-most-recent dbspace backups. Universal 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)

Universal 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 42-3 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

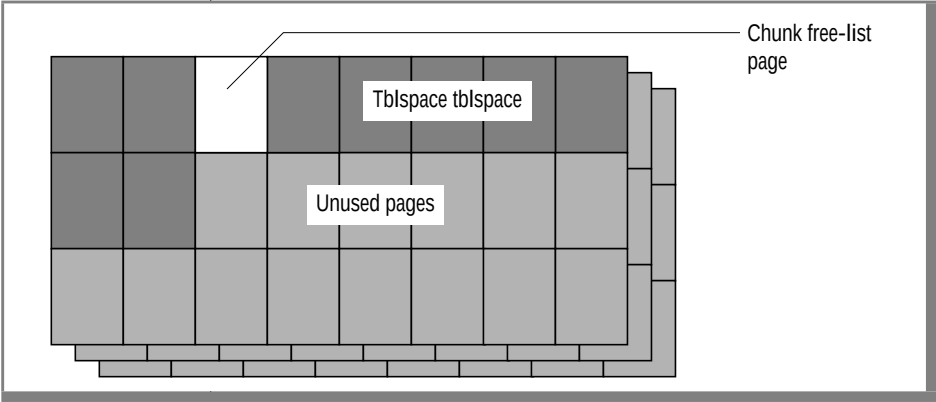


Figure 42-3
*Initial Chunk of
Regular Dbspace*

Figure 42-5 illustrates the mirror-chunk structure as it appears after the chunk is created.

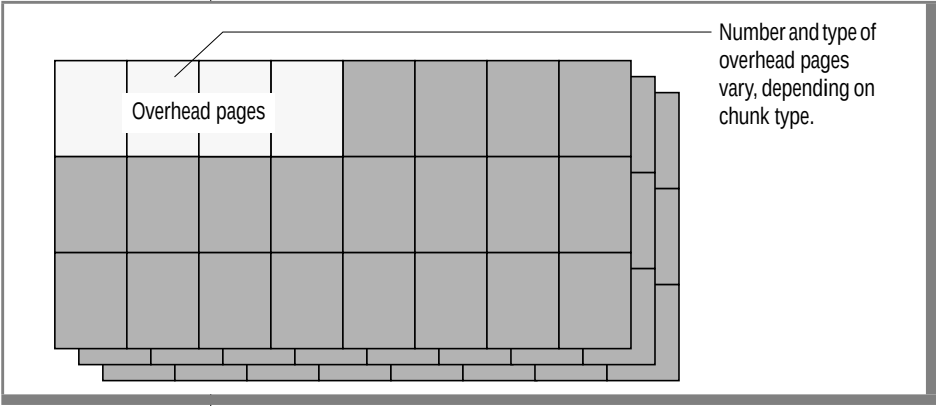


Figure 42-5
Mirror-Chunk Structure

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 27-11.](#)

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 42-6 illustrates the location of the free-list page.

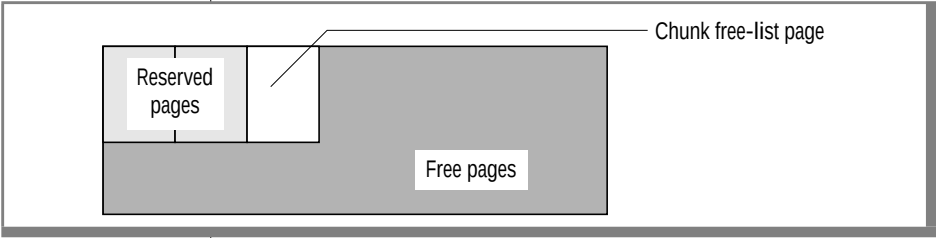


Figure 42-6
Free-List Page

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 42-7 illustrates a sample listing from a chunk free-list page.

Figure 42-7
Sample Listing from a Chunk Free-List Page

Chunk Offset	Number of Free Pages
14	28
123	36
208	52

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 Tblspace Tblspace

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 42-8 illustrates the location of the tblspace tblspace.

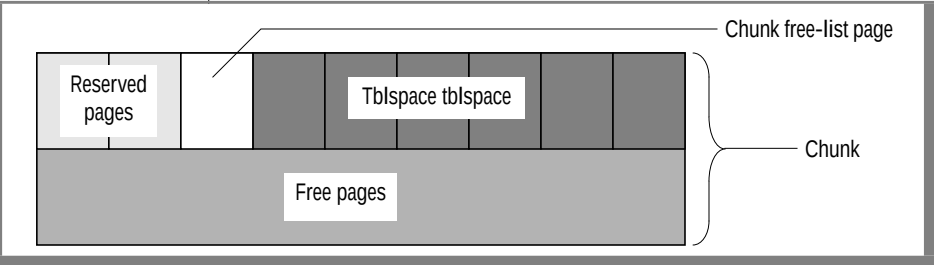


Figure 42-8
Tblspace Tblspace

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, <i>database.owner.tablename</i>

(1 of 2)

Component	Description
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.

(2 of 2)

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 42-9 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 42-9
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 42-10 shows that the **account** table from Figure 42-9 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 42-10
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 (1-1/2 bytes) indicate the dbspace number where the tblspace resides. The least-significant 20 bits (2-1/2 bytes) indicate the logical page number where the tblspace is described. Figure 42-11 illustrates the elements of a tblspace number. Compare this format to the actual **partnum** values in [Figure 42-9 on page 42-20](#).

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.)

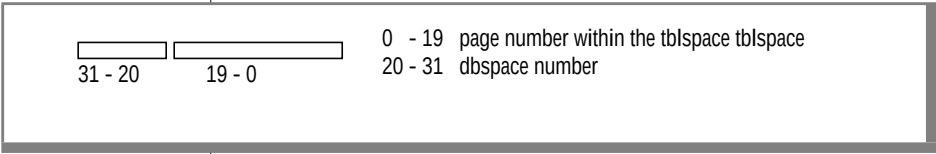


Figure 42-11
The Tblspace
Number with
Dbspace Number
and Page Number

Tblspace Tblspace 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 Universal Server attempts to create a table, but the tblspace tblspace is full, Universal 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.

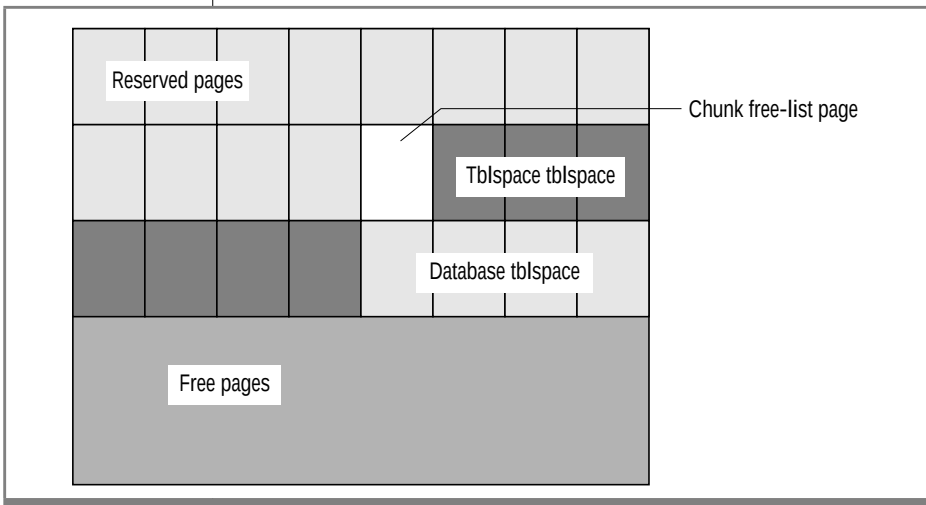
Tblspace Tblspace Bit-Map Page

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 Universal Server. Figure 42-12 illustrates the location of the database tblspace.

Figure 42-12
*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.

Universal 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 42-13.

Figure 42-13
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 42-14. 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 42-14
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	Simple-large-object page, can accept whole-page segments
0110	Simple-large-object page, room for partial-page segments
1010	Simple-large-object page, room for small segments
1110	Simple-large-object 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 Universal 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, refer to the [*INFORMIX-Universal Server 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.

Tbspaces that hold *index fragments* follow different rules for extent size. Universal Server bases the extent size for these tablespaces on the extent size for the corresponding table fragment. Universal Server uses the ratio of the row size to index key size to assign an appropriate extent size for the detached index tspace. (For information on fragmented indexes, refer to [“Fragmentation of Table Indexes” on page 16-17.](#))

Tbspaces that hold *index fragments* follow different rules for extent size. Universal Server bases the extent size for these tablespaces on the extent size for the corresponding table fragment. Universal Server uses the ratio of the row size to index key size to assign an appropriate extent size for the index tspace. (For information on fragmented indexes, refer to [“Fragmentation of Table Indexes” on page 16-17.](#))

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 Universal Server page, the bit map is a four-bit bit map; otherwise, the bit map is a two-bit bit map.

- Blobpages

Blobpages contain simple large objects that are stored with the data rows in the dbspace. Simple large objects that reside in a blob space are stored in blobpages, a structure that is completely different than the structure of a dbspace blobpage.

- Free pages

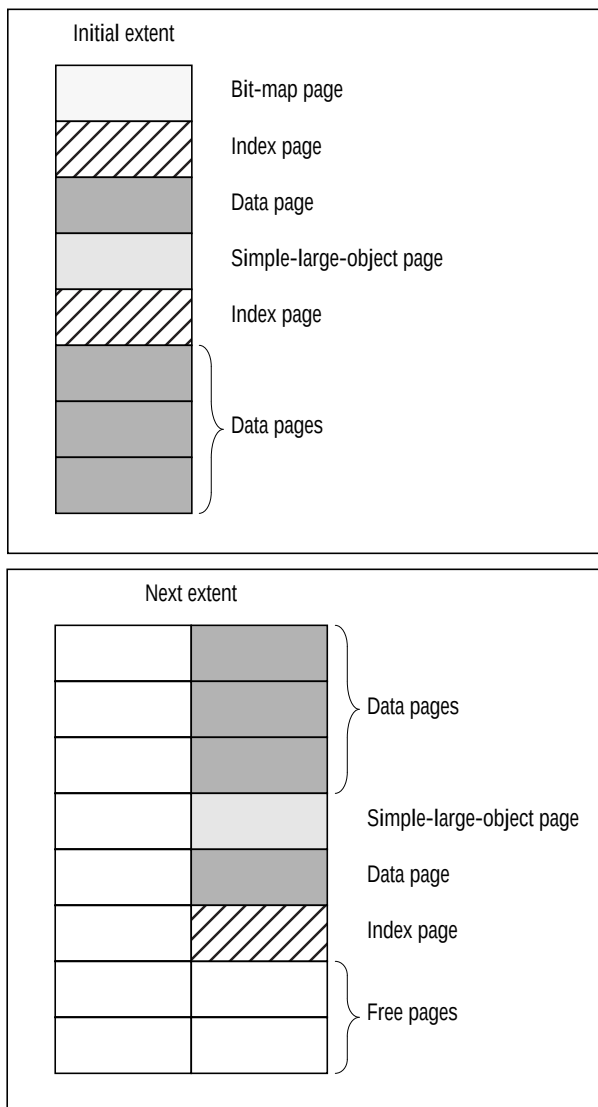
Free pages are pages in the extent that are allocated for tblspace use, but whose function has not yet been defined. Free pages can be used to store any kind of information: data, index, simple large object, or bit map.



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, refer to “[Fragmentation of Table Indexes](#)” on [page 16-17](#).)*

[Figure 42-15 on page 42-30](#) 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 42-15
Extent Structure of a
Table



Next-Extent Allocation

After the initial extent fills, Universal Server attempts to allocate another extent of contiguous disk space. The procedure that Universal Server follows is referred to as next-extent allocation.

Extents for a tblspace are tracked as one component of the 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 entry.

Next-Extent Size

The number of kilobytes that Universal 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 42-16 on page 42-33](#).

Extent Size Doubles Every 16 Extents

If a tblspace already has 16 extents allocated, Universal 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, Universal Server allocates the first 16 extents at a size of 20 kilobytes each. Universal Server allocates extents 17 to 32 at 40 kilobytes each, extents 33 to 48 at 80 kilobytes each, and so on.

What If Universal Server Cannot Find Contiguous Space?

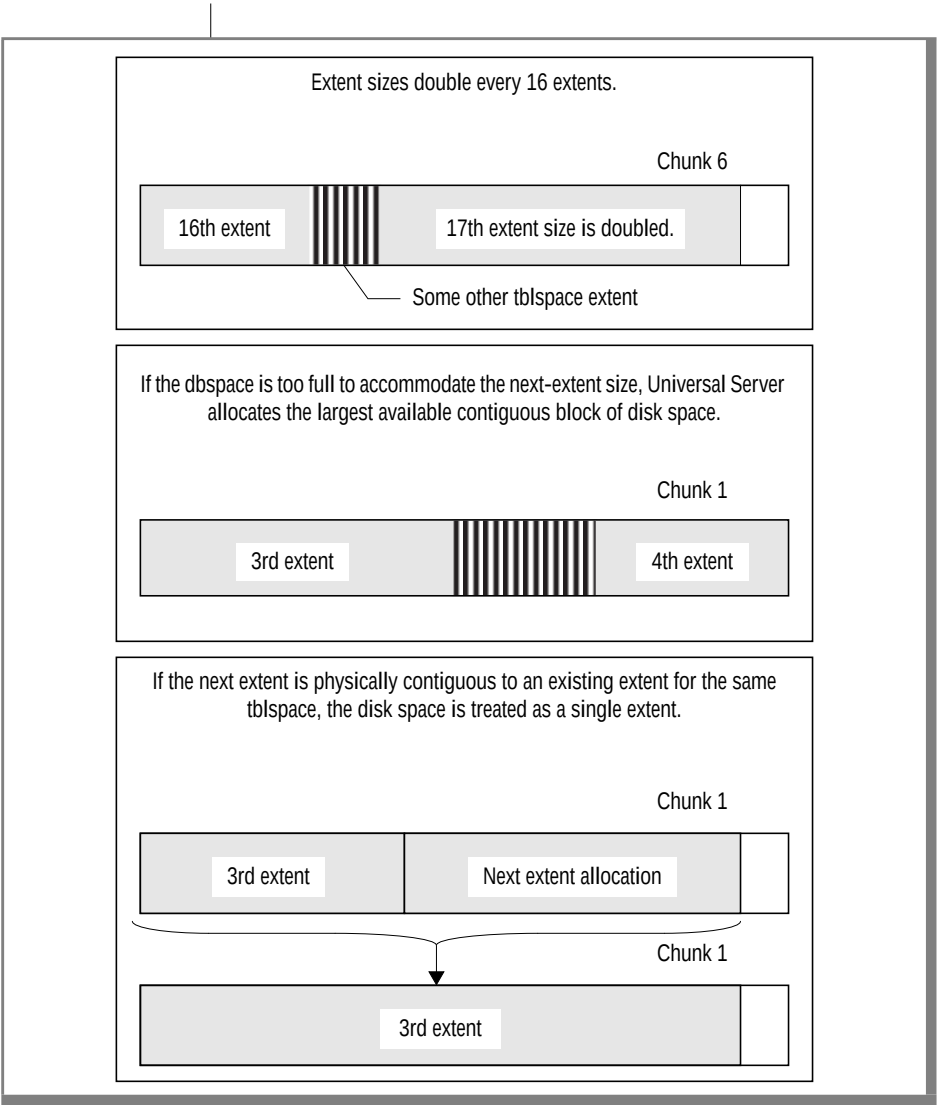
If Universal Server cannot find available contiguous space in the first chunk equal to the size specified for the next extent, it extends the search into the next chunk in the dbspace. Extents are not allowed to span chunks.

If Universal 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, Universal Server allocates the disk space but does not consider the new allocation as a separate extent. Instead, Universal Server extends the size of the existing contiguous extent. Thereafter, all Universal Server disk-space reports reflect the allocation as an extension of the existing extent. That is, the number of extents reported by Universal Server 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 42-16 on page 42-33](#) illustrates extent-allocation strategies.

Figure 42-16
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 the [INFORMIX-Universal Server Performance Guide](#).

Structure and Storage of a Dbspace Page

This section describes how a dbspace page is structured and how Universal Server uses this structure to store data in a dbspace.

Structure of a Dbspace Page

The basic unit of Universal Server I/O is a page. Page size might vary among computers. You cannot modify the page size.

Page Elements

Universal 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 Universal Server adhere to a similar structure, although the function of the page can alter slightly the size of structures within the page. Figure 42-17 illustrates the following three structures that appear on every Universal Server page:

- Page header (24 bytes, including one 4-byte time stamp)
- Page-ending time stamp (4 bytes)
- Slot table (4 bytes per entry)

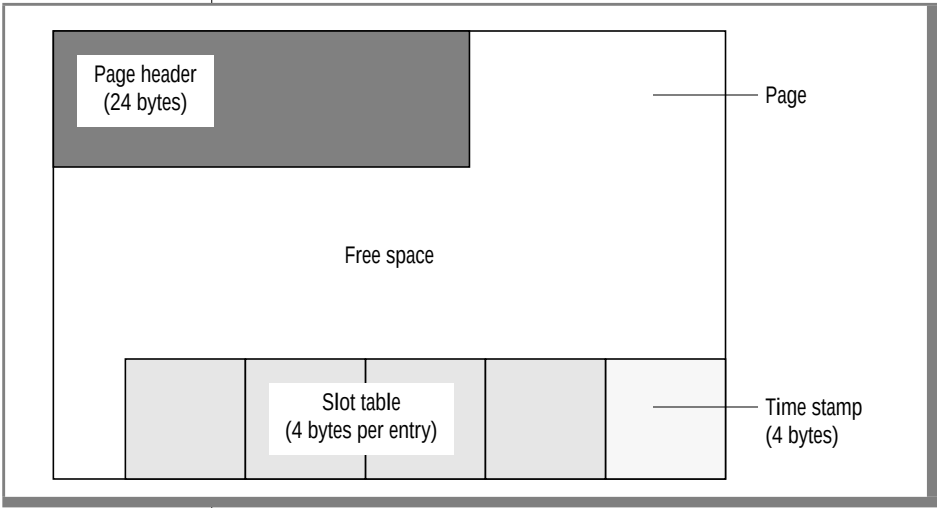


Figure 42-17
Dbspace Page Structure

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 Universal 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 that can be stored in a single data page.

Slot Table Stays Accurate Even If Row Moves

The slot table is the only Universal Server structure that points to a specific location in a data page. For this reason, Universal 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 Universal 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.
- Simple large objects are not stored within the data row. Instead, the data row contains a 56-byte descriptor that points to the location of the simple large object. (The descriptor can point to either a dbspace simple-large-object page or a blob space blobpage. If you are using INFORMIX-OnLine/Optical, 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, refer to the [INFORMIX-Universal Server 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 42-39](#).

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 42-18 illustrates the rowid format.

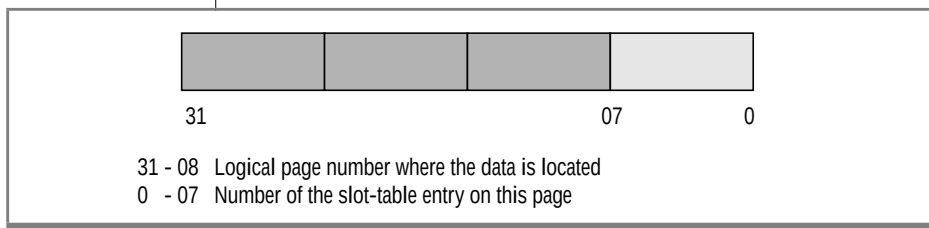


Figure 42-18
Rowid Format

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, Universal 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 [“Creating a Rowid Column” on page 16-21](#). If user applications attempt to reference a rowid in a fragmented table that does not contain a rowid that you explicitly created, Universal 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, Universal Server uses an index to map the rowid to a physical location.

When Universal 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, Universal 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, refer to 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 Universal 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 42-19 illustrates data storage when rows are less than a page.

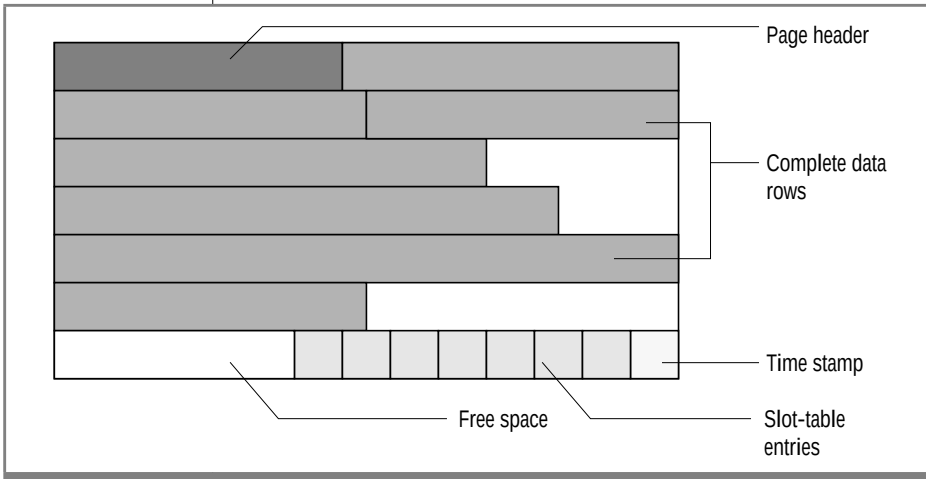


Figure 42-19
*Rows Shorter Than
a Page*

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 42-20 illustrates the concepts of home page, big-remainder page, and remainder page.

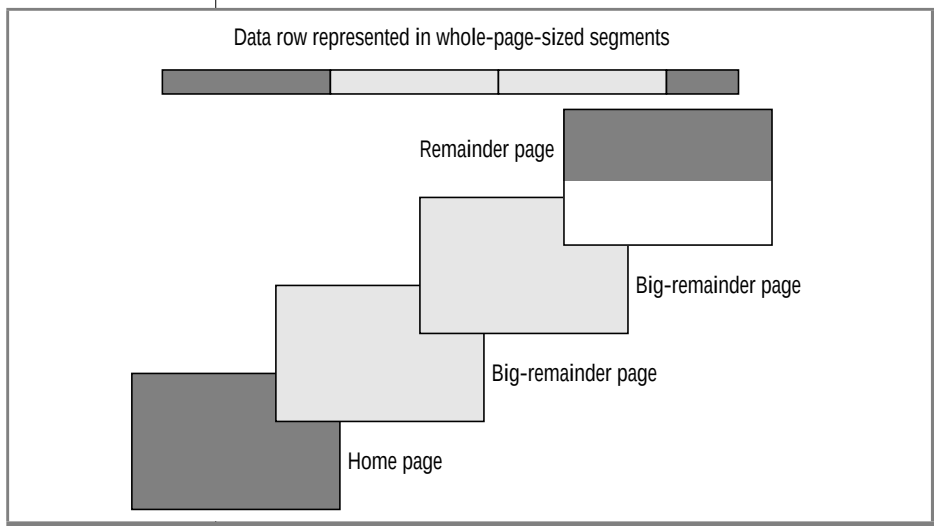


Figure 42-20
Remainder Pages

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.

Figure 42-21 illustrates data storage for rows that are longer than two pages

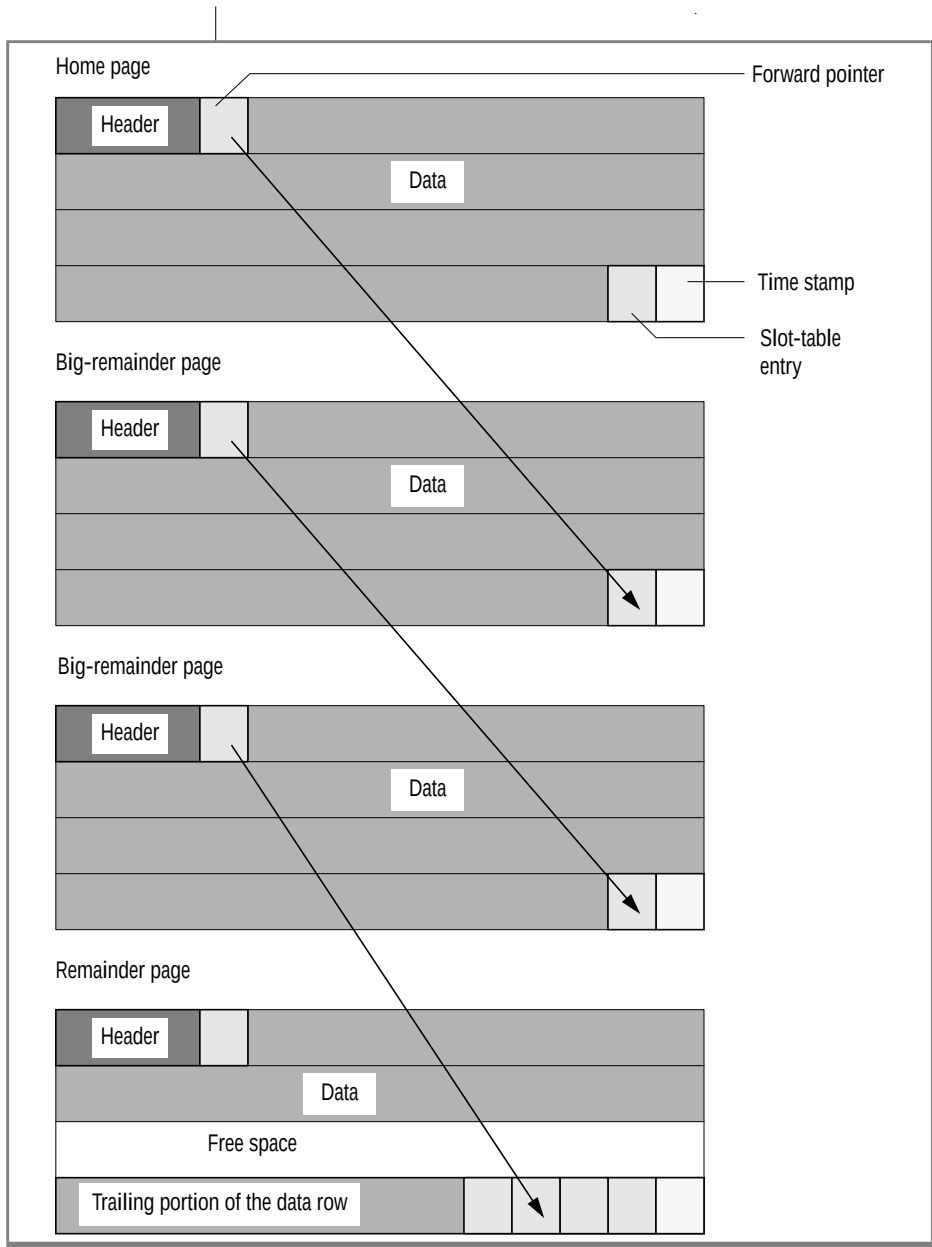


Figure 42-21
Rows Longer Than
Two Pages

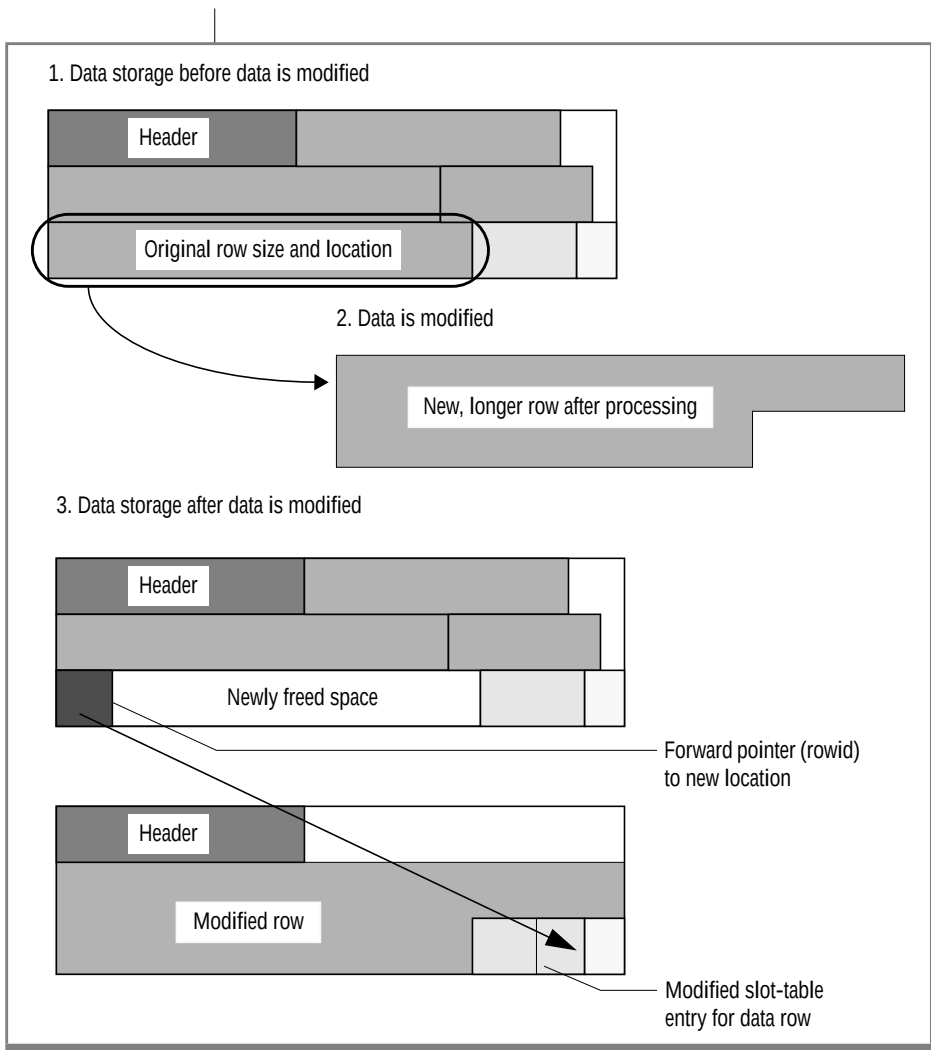
What Happens When a Row Is Modified?

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, Universal 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 42-22 on page 42-45](#) illustrates data storage if the updated row is too large for the home page but shorter than a whole page.

Figure 42-22
Updated Rows



What Happens When the Updated Row Is Longer Than a Whole Page?

If the modified data row is longer than a page, Universal Server first begins to divide the data into whole-page segments, starting from the *tail* end of the row. Universal Server then attempts 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, Universal Server divides the page into whole-page segments again, but this time beginning with the leading end of the row. Universal 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 42-23 on page 42-47](#) illustrates storage of an updated row that is longer than a whole page.

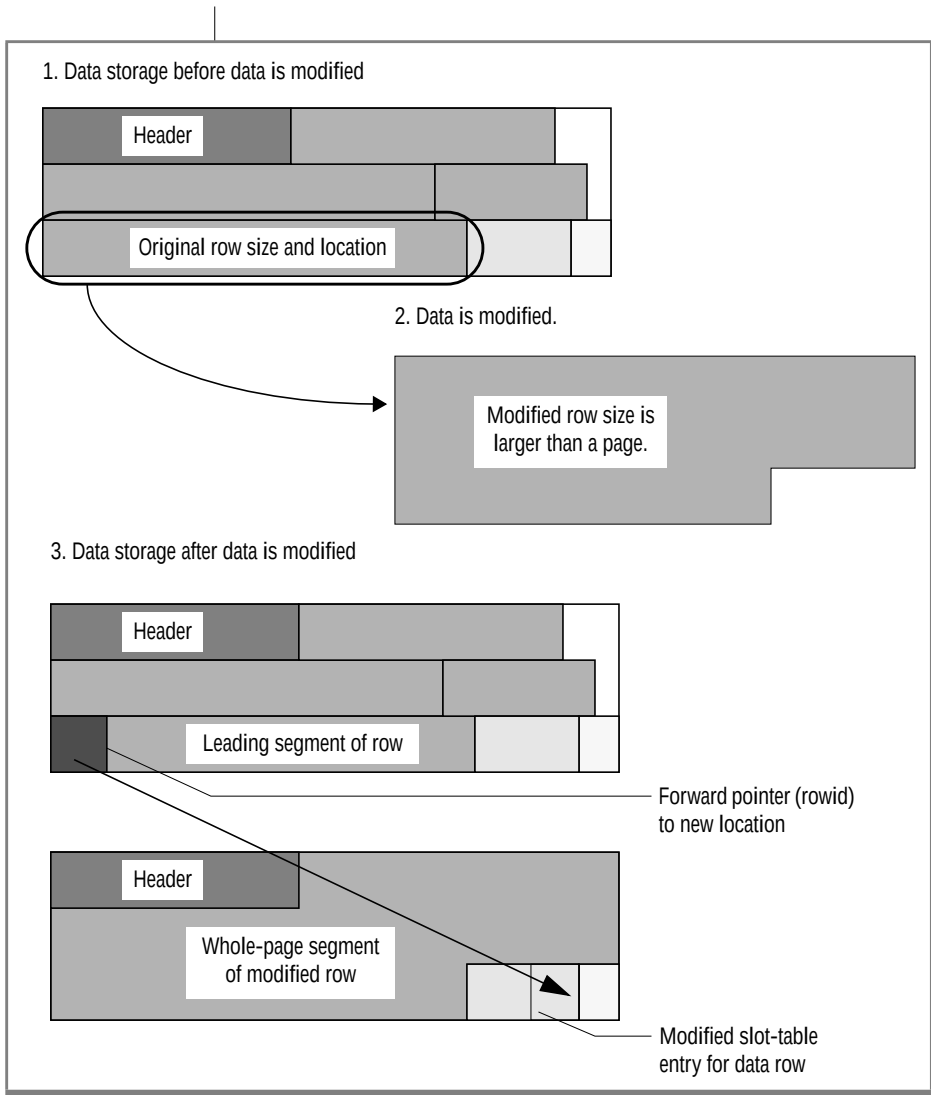
Page Compression

Over time, the free space on a page can become fragmented. When Universal 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, Universal Server checks if the page contains adequate contiguous free space to hold the row (or row portion). If the free space is not contiguous, Universal 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 42-23
Updated Rows That
Span Pages



Structure of B-Tree Index Pages

This section provides general information about the structure of Universal Server 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 Universal Server uses to create and fill indexes is then explained. The latter part of the section describes the physical layout of an index page.

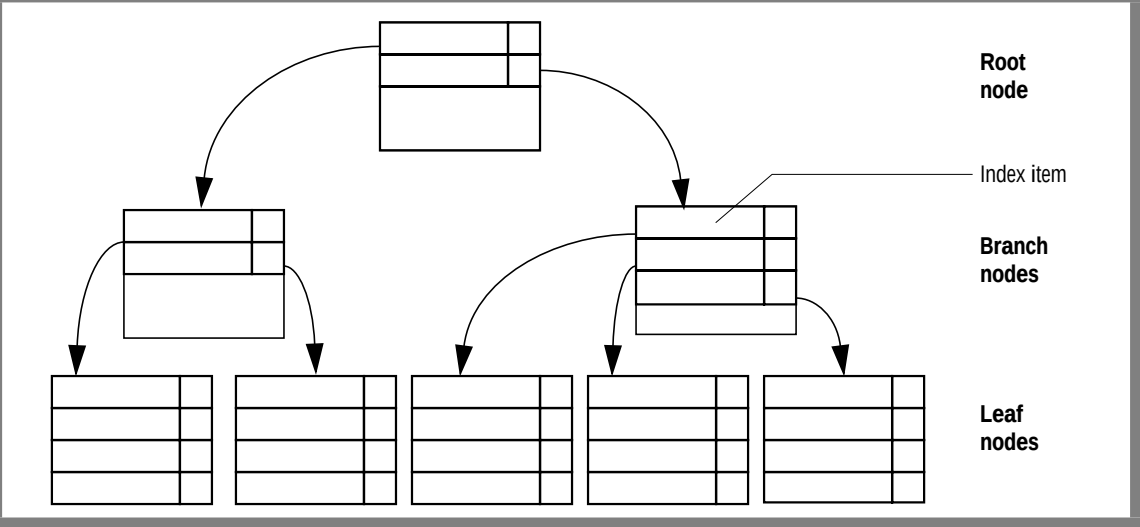
Definition of Terms in B-Tree Indexing

Universal Server uses a B-tree structure for organizing index information. As shown in [Figure 42-24 on page 42-49](#), 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 42-24
Full B-Tree Structure



What Is an Index Item?

The fundamental unit of indexes is the *index item*. As shown in Figure 42-25, 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 42-25
Logical Depiction of an Index Item

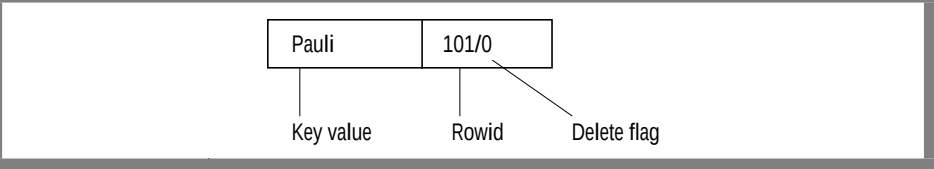


Figure 42-25 on page 42-49 illustrates an example of an index item. The index item illustrated is row 101 in the **customer** table. The **lname** value of row 101 is Paul i. The delete flag is set to 0, indicating that the row still exists. The index item for this **lname** value is composed of a key value, Paul i, 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 42-26 shows a logical depiction of an index node.

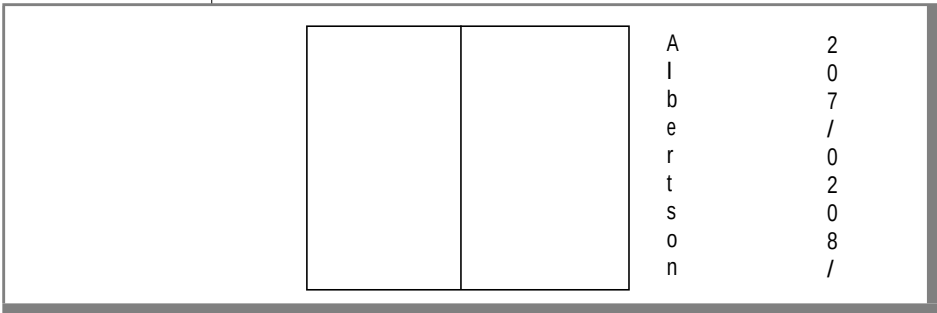


Figure 42-26
An Index Node

B-tree structures contain the following three types of nodes: root, branch, and leaf nodes. When an index is fully developed (see Figure 42-24 on page 42-49), 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 of Indexes” on page 42-55.

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 of Indexes” on page 42-55.

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 42-27 illustrates how a root node appears before it fills.

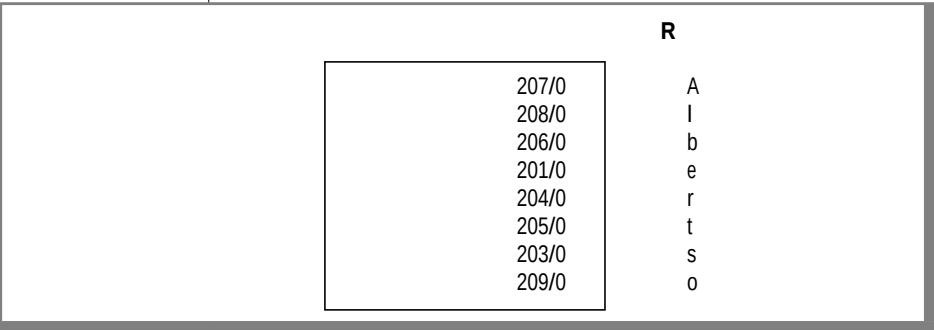


Figure 42-27
Root Node

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 42-28.

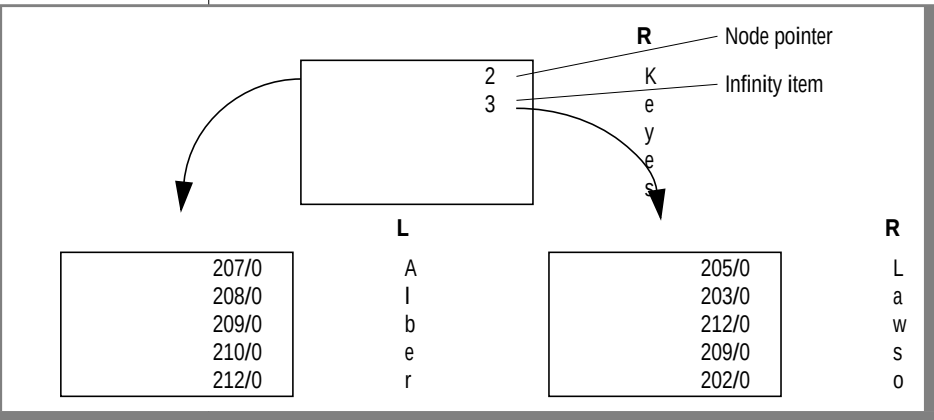


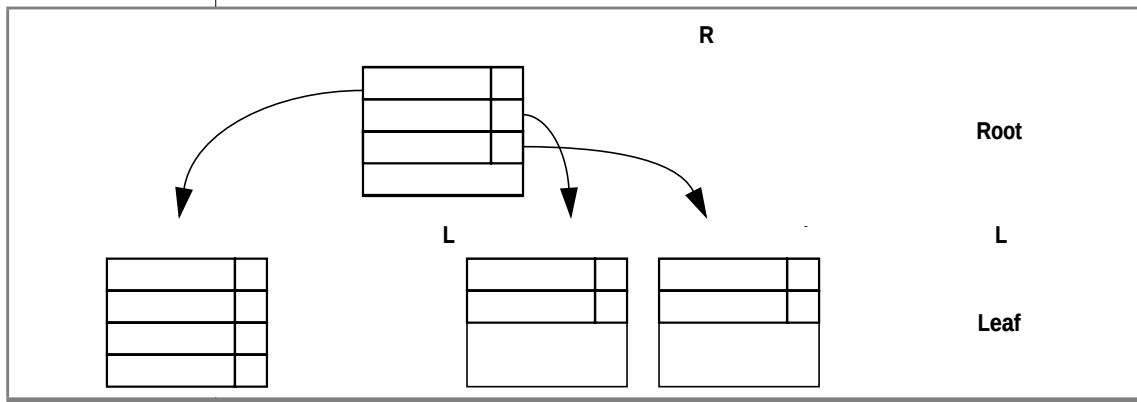
Figure 42-28
Two Leaf Nodes
Created After a Root
Node Becomes Full

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 Universal 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 42-28 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 42-29.

Figure 42-29
Leaf Node 4 Created After Leaf Node 3 Fills

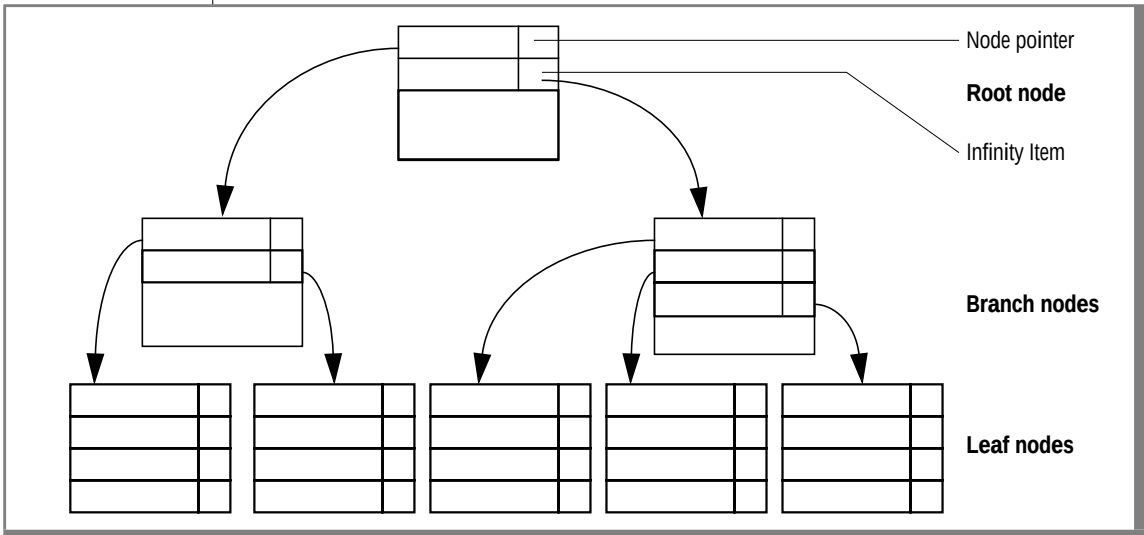


Creation of Branch Nodes

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 42-30 illustrates this structure.

Figure 42-30
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 42-31, 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.

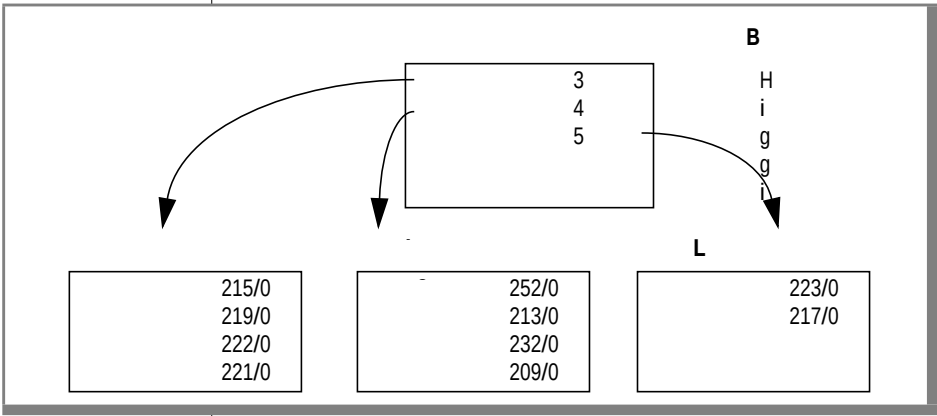


Figure 42-31
Typical Contents of
a Branch Node

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 42-32 on page 42-55](#).

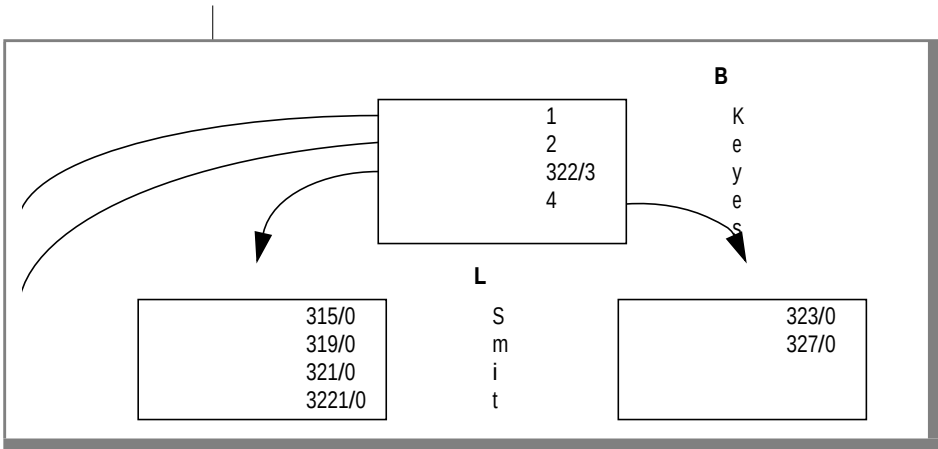


Figure 42-32
Leaf Nodes 3 and 4

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 of Indexes

This section explains the physical-storage format of index pages.

Physical Storage Format of Index Pages

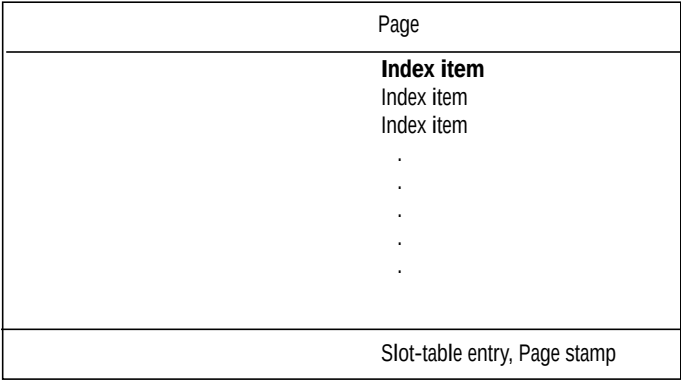
An empty node is a 2- or 4-kilobyte area with a page header at the top and a stamp and empty slot-table at the bottom (see Figure 42-33). The page header contains the following information:

- 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 42-33
*Physical Format of
Newly Created Index
Page*



The slot table grows upward from the bottom of the page as Universal 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, refer to 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 ONCONFIG parameter FILLFACTOR 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 42-34 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.

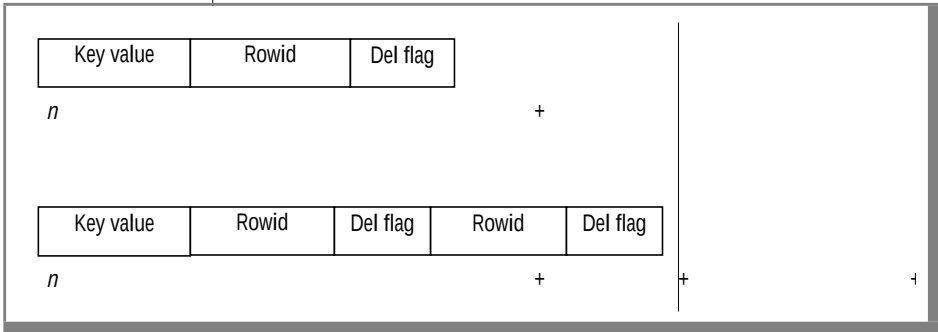


Figure 42-34
Index-Item Formats

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 42-34 illustrates the structure of an index item.

Functional Indexes

A *functional index* is one in which all keys derive from the results of a function. If you have a column of pictures, for example, and you have a function that identifies the predominant color, you could create an index on the result of the function. Such an index would enable you to quickly retrieve all pictures having the same predominant color, without re-executing the function.

A functional index uses the same B-tree structure as any other B-tree index. The only difference is that the determining function is applied during an insert or an update whenever the column that is the argument to the function changes. For more information on the nature of functional indexes, refer to the [INFORMIX-Universal Server Performance Guide](#).

Creating a Functional Index

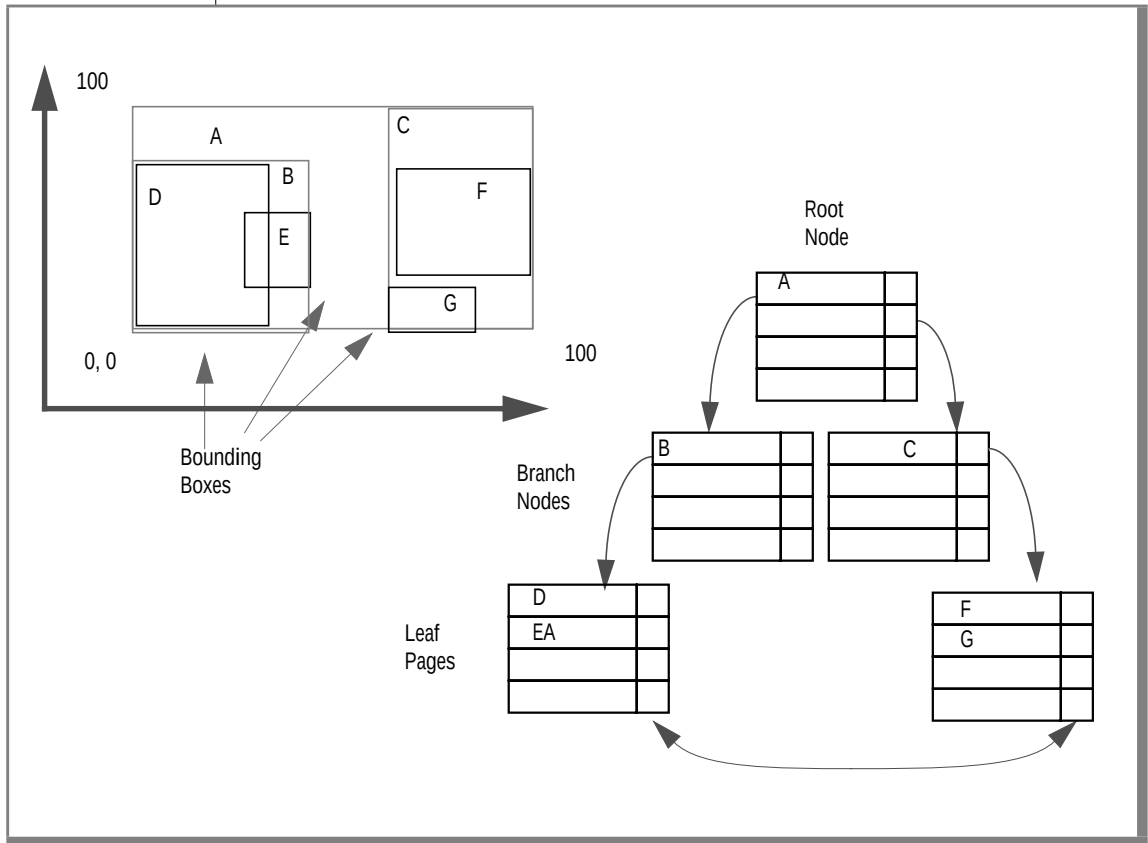
You create a functional index using the CREATE FUNCTION and CREATE INDEX statements. For more information on these statements, refer to the [Informix Guide to SQL: Syntax](#).

Structure of R-Tree Index Pages

An index structure that relies on one-dimensional ordering of key values does not work for spatial data such as two dimensional geometric shapes like circles, squares and triangles. Efficient retrieval of spatial data, such as the data used in geographic information systems (GIS) and computer-aided design (CAD) applications, requires an access method that handles multi-dimensional data. Universal Server implements an R-tree index to access spatial data efficiently. You should create an R-tree index on spatial data columns that you are likely to reference in a search condition. The Spatial DataBlade module, which you can purchase as an add-on to Universal Server, uses the R-tree index.

An R-tree is a tree-based access method, similar in some ways to a B-tree. Internal nodes store pointers to lower-level nodes. In an R-tree, however, the root node and the branch nodes store the coordinates of *bounding boxes*. The bounding box is a boundary that encompasses subnodes and the data objects that the branch indexes. The leaves of the tree, or the data pages, contain the keys, or coordinates for spatial objects, and pointers to their rows in the table. [Figure 42-35 on page 42-61](#) illustrates the basic concept of an R-Tree.

Figure 42-35
R-Tree Index for Four Rectangles



Each rectangle shown in Figure 42-35 is stored as two pairs of coordinates, one pair representing the lower left corner and one pair representing the upper right corner. A bounding box, then, can also be thought of as a rectangle and defined as two pairs of coordinates which contain the objects within it. For example, assume that rectangles D and E have the following coordinates:

- D=16, 20 and 40, 60
- E=35, 35 and 52, 45

Then the coordinates of bounding box B would be as follows:

B=16,20 and 52,60

Universal Server expands bounding boxes in response to insertions. Searches and insertions descend the tree, checking the bounding boxes at each node to see if they include an object that the query is seeking. When a matching subtree is found, Universal Server descends the subtree recursively.

When Universal Server descends to a leaf node on a search, it examines each entry on the leaf page to see whether it satisfies the search criteria.

For an insert, Universal Server examines subtrees to determine which one will need to expand the least amount to add the new object. If more than one box qualifies, Universal Server chooses the first one it finds. When Universal Server reaches the leaf node, it creates a new entry and stores the object.

For more information on the structure of an R-tree index, and for information on estimating the size of an R-tree index, refer to the [INFORMIX-Universal Server Performance Guide](#).

Blobspace Structure and Storage

This section explains the structures and storage techniques that Universal Server uses to store simple large objects in a blobspace.

Structure of a Blobspace

When you create a blobspace, you can specify the effective size of the simple large object-holding pages, 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 Universal Server Page Size](#)” on [page 15-15](#).) 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 simple-large-object 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, simple large objects are stored in one or more blobpages, but simple large objects do not share blobpages. Simple-large-object storage is most efficient when the simple large object 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 42-36 illustrates the chunk structure of a blobspace as it appears immediately after the blobspace is created.

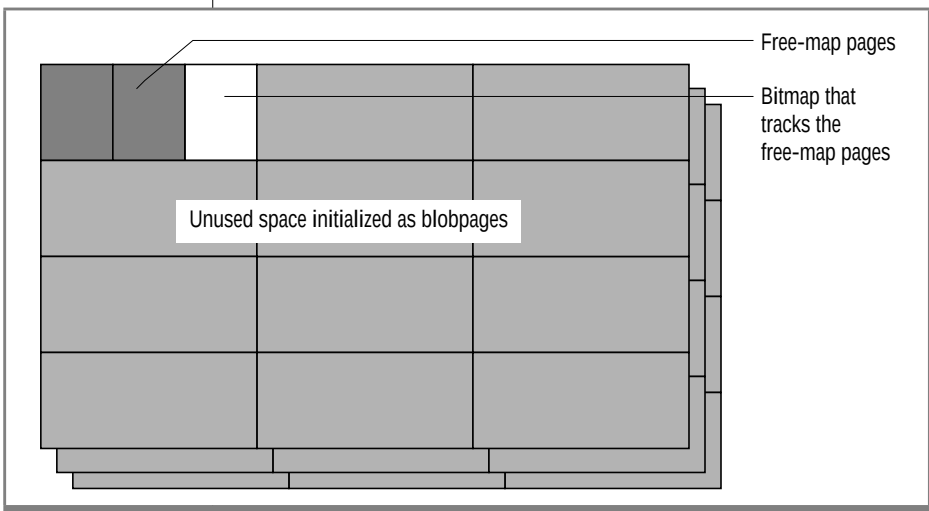


Figure 42-36
*Structure of a Chunk
in a Blobspace*

Simple-Large-Object Storage and the Simple-Large-Object Descriptor

Data rows that include simple-large-object data do not include the simple-large-object data in the row itself. Instead, the data row contains a 56-byte simple-large-object descriptor that includes a forward pointer (rowid) to the location where the first segment of simple-large-object data is stored. The descriptor can point to a simple-large-object page (if the simple large object is stored in a dbspace), a blobpage (if the simple large object is stored in a blobspace), or an optical platter (if you are using INFORMIX-OnLine / Optical).

Figure 42-37 illustrates the structure of the 56-byte simple-large-object descriptor.

Figure 42-37
Structure of the Simple-Large-Object Descriptor

```
typedef struct tblob
{
    short    tb_fd;        /* simple large object descriptor (must be first) */
    short    tb_coloff;    /* simple large object column offset in row */
    long     tb_tblspace; /* simple large object table space */
    long     tb_end;       /* ending byte: 0 for end of simple large object */
    long     tb_size;      /* Size of simple large object */
    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 simple large object stamp */
    short    tb_sockid;    /* socket id of remote simple large object */
    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 Simple Large Objects Created?

When a row that contains simple-large-object data is to be inserted, the simple large objects are created first. After the simple large objects are written to disk (or optical medium), the row is updated with the simple-large-object descriptor and inserted.

Are Simple Large Objects Modified?

Simple large objects are never modified. Simple large objects can only be inserted or deleted. Deleting a simple large object means that Universal Server frees the space consumed by the deleted simple large object for reuse.

When simple-large-object data is updated, a new simple large object is created, and the data row is updated with the new simple-large-object descriptor. The old image of the row contains the descriptor that points to the obsolete simple-large-object value. The space consumed by the obsolete simple large object is freed for reuse after the update is committed. Simple large objects are automatically deleted if the rows containing their simple-large-object descriptors are deleted. (Blobpages that stored a deleted simple large object are not available for reuse until the logical log that contains the original INSERT record for the deleted simple large object is backed up. For more information, see [“Backing Up Logical-Log Files to Free Blobpages” on page 22-24](#))

What Limits the Simple-Large-Object Size?

The largest simple large object that the simple-large-object 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 simple large object in the simple-large-object descriptor.

Structure of a Dbspace Blobpage

Simple-large-object 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 simple-large-object data in the data area.

Simple large objects can share dbspace blobpages if more than one simple large object can fit on a single page, or if more than one trailing portion of a simple large 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, refer to the [INFORMIX-Universal Server Performance Guide](#).

Each segment of simple large object 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 simple-large-object time stamp for this simple-large-object segment (required)
- A 4-byte forward pointer (rowid) to the next portion of the simple-large-object segment, if one exists (optional)
- A 4-byte simple-large-object time stamp stored with the forward pointer to the next portion of the simple-large-object segment (required if a forward pointer exists)

Figure 42-38 illustrates simple-large-object data storage in a dbspace.

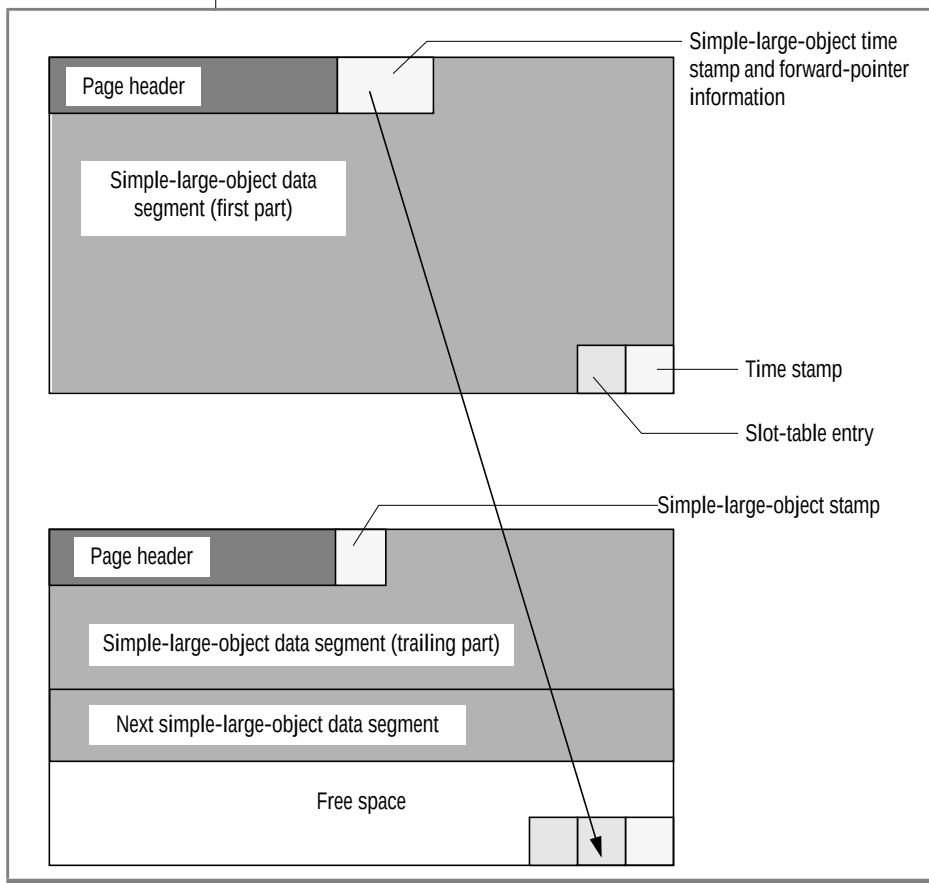


Figure 42-38
Dbspace Blobpage

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 simple-large-object creation. When a blobpage is allocated, the free-map entry for that page is updated. All entries for a single simple large 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 simple-large-object logging.)
- The second word contains the tblspace number associated with the simple large 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 simple-large-object data. Blobpage size is specified by the database server administrator who creates the blobpage. Blobpage size is specified as a multiple of the page size.

Structure of a Blobspace Blobpage

The storage strategy used to store simple large objects in a blobpage differs from the dbspace storage strategy. The database server does not combine whole simple large objects or portions of a simple large object on a single blobpage blobpage. For example, if blobpage blobpages are 24 kilobytes each, a simple large 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 simple-large-object data, and a page-ending time stamp. The blobpage header includes, among other information, the page-header time stamp and the simple-large-object time stamp associated with the forward pointer in the data row. If a simple large object is stored on more than one blobpage, a forward pointer to the next blobpage and another simple-large-object time stamp are also included in the blobpage header.

Figure 42-39 illustrates the structure of a blobpage.

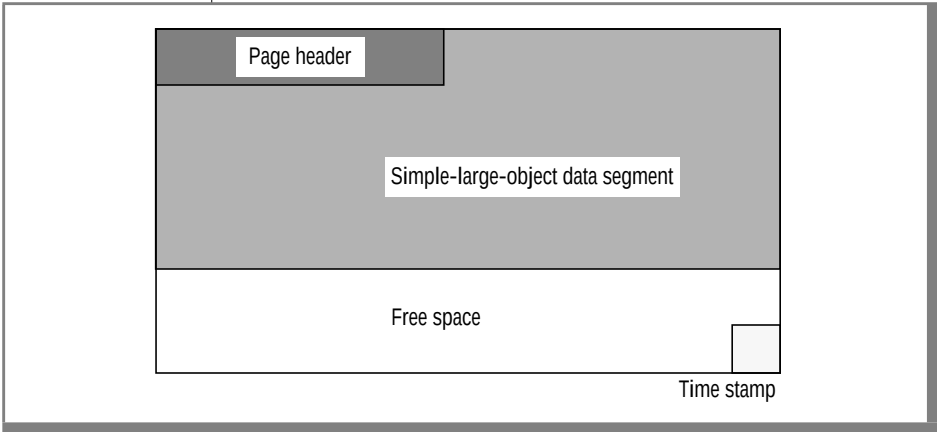


Figure 42-39
Blobspace Blobpage

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 simple-large-object data and an associated simple-large-object time stamp, if a next segment exists

Otherwise, only the current page number appears, indicating this is the last page.

- A simple-large-object time stamp that describes the last time this page was allocated (when simple-large-object 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 42-40 on page 42-70](#) illustrates the different locations of the two pairs of time stamps that appear on the blobpage blobpage.

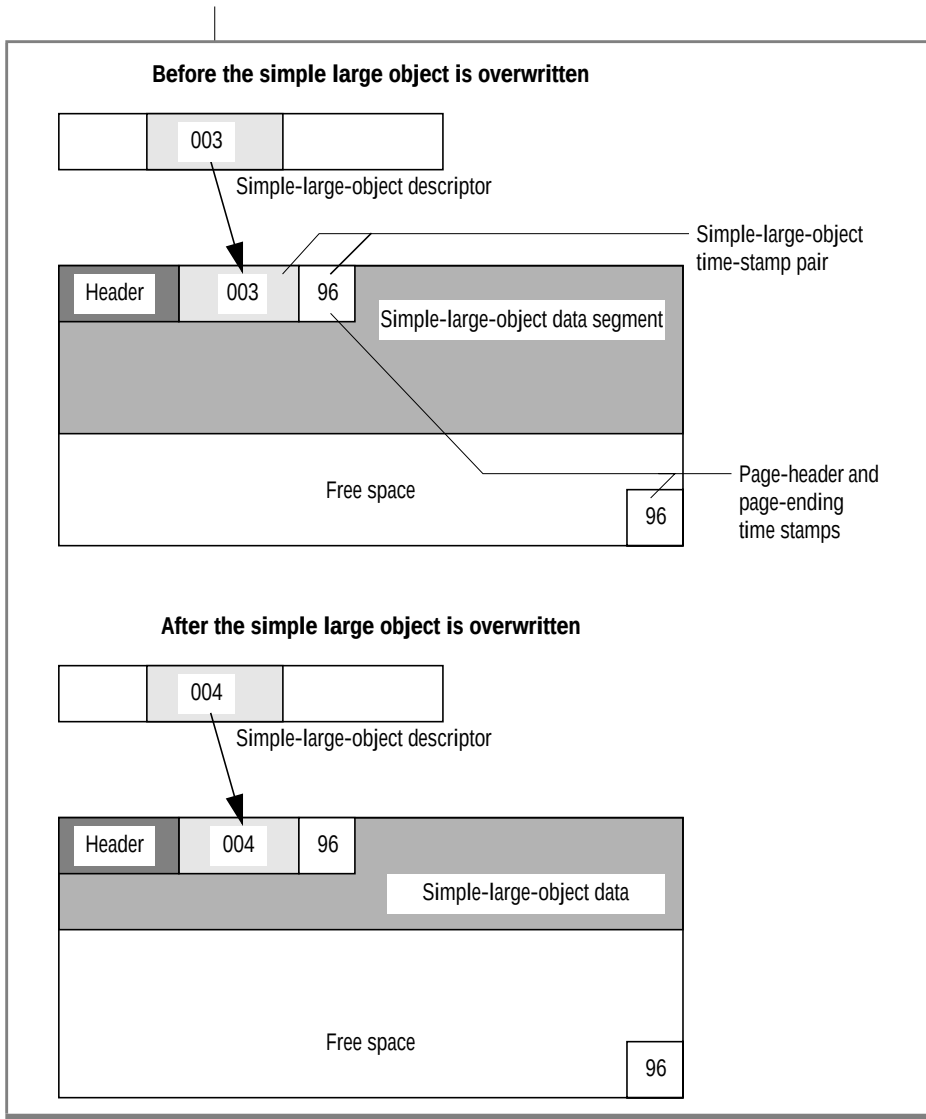


Figure 42-40
Blobpage Time Stamps

The simple-large-object 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.

Sbospace Structure

An sbospace is similar in concept to the current blobspace. When an sbospace is created in a database, it contains the following structures:

- Sbospace chunk header pages
- An sbospace metadata area
- A user data area

For best performance, we recommend that the metadata area be located in the middle of the sbospace. Figure 42-45 illustrates the chunk structure of an sbospace as it appears immediately after the sbospace is created.

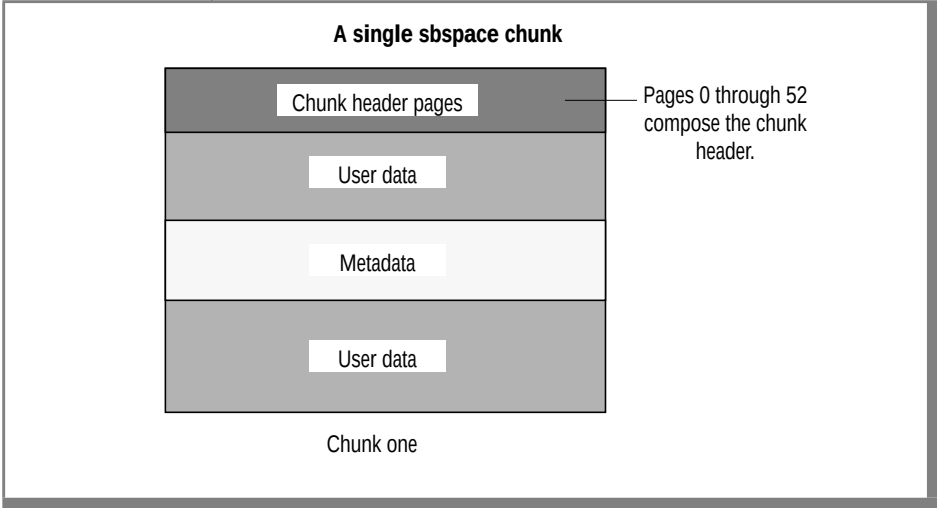


Figure 42-41
*A Single Sbospace
Chunk*

Since the chunk in Figure 42-45 is the first in the sbospace, its chunk header contains additional information not found in the chunk header of chunks added to the sbospace subsequently. The chunk header pages for chunk one, contain information which tracks various information about chunk one and for all chunks added to the sbospace thereafter. Figure 42-42 illustrates the chunk header structure for the first chunk in an sbospace.

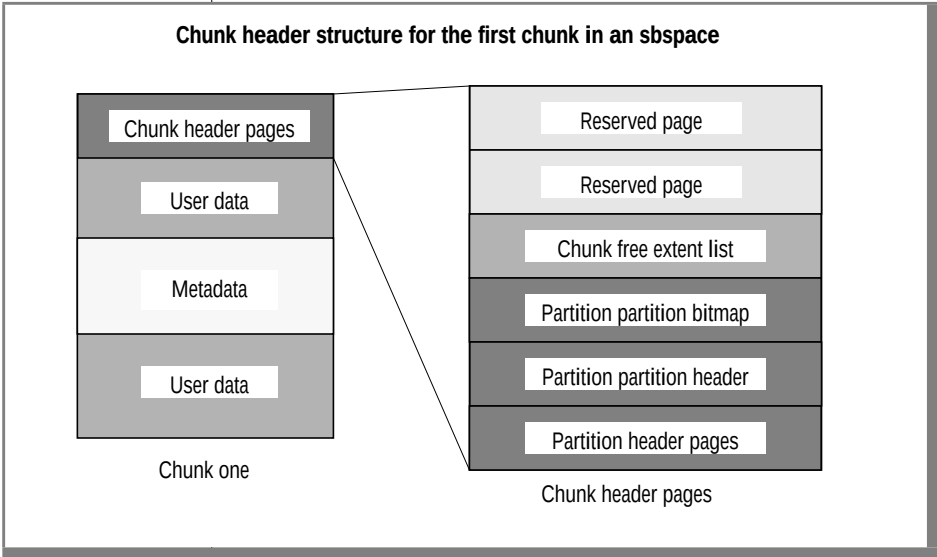


Figure 42-42
*Chunk Header
Structure for the
First Chunk in a
Sbospace*

The chunk header pages of the first chunk in a sbospace, along with all metadata areas in all additional chunks, are managed as an entirely separate dbospace within the sbospace. The Chunk free extent list tracks extents of the associated chunk only. The partition partition bitmap is the standard bitmap of free pages within the partition partition. The partition partition header is the required partition header for the partition partition. The partition partition pages (that is, the partition header pages) are stored in pages five through fifty-two. If more space is required, it is allocated from chunk one metadata. When adding additional chunks to an sbospace, the chunk header for the additional chunks includes only the two reserved pages and the Chunk free extent list.

Metadata Area Structure

As with the chunk header pages, there are four areas exclusive to the first chunk in a sbospace, the sbospace descriptor partition, the chunk adjunct partition, and the level one and two archive partitions. The partition header section contains a partition header for each of these partitions (notably excluding the partition partition). Figure 42-43 shows the layout of the metadata in the single-chunk sbospace.

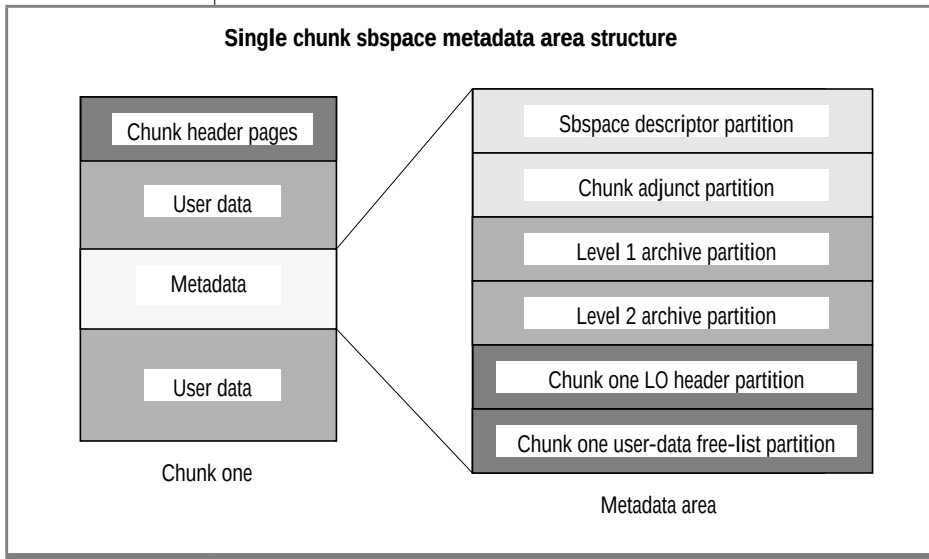


Figure 42-43
*Single Chunk
Sbospace Metadata
Area Structure*

Sbpage Structure

Each sbpage is composed of three elements, a sbpage header; the actual user data itself; and a sbpage trailer. Figure 42-44 shows the structure of a sbpage.

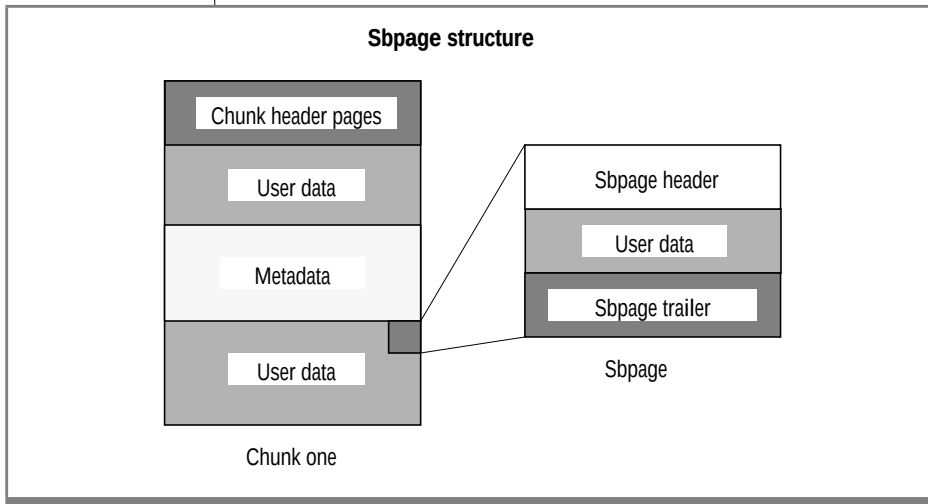


Figure 42-44
Sbpage Structure

The sbpage header is composed of the standard RSAM page header. The sbpage trailer is used to detect an incomplete write on the page and to detect page corruption.

A Multiple Chunk Sbspace

The figure below illustrates a possible configuration for a three chunk sbspace. In this example, chunk two contains no metadata of its own. Metadata information for chunk two is stored in the metadata area of chunk one.

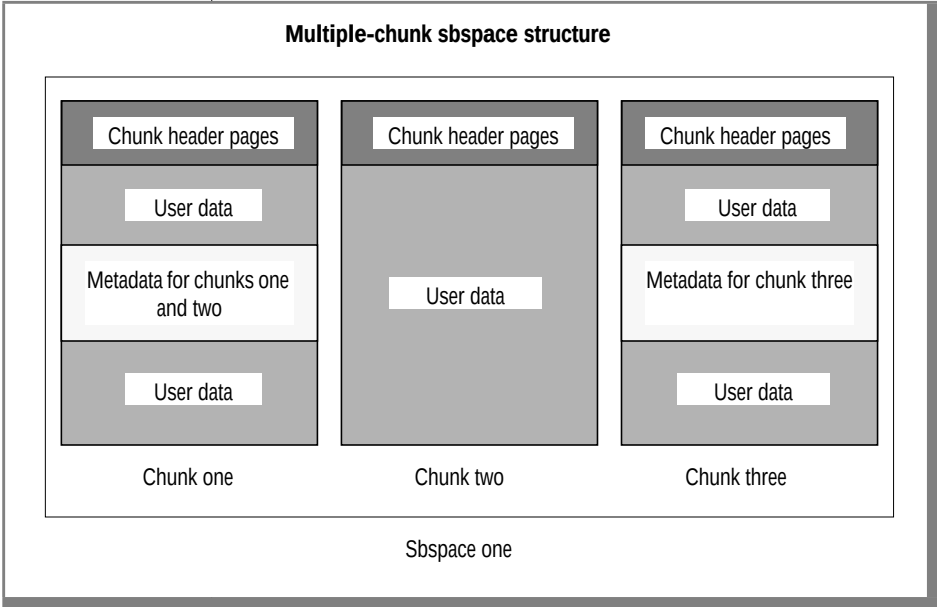


Figure 42-45
*Multiple-Chunk
Sbspace Structure*

The user data areas in chunk one of the example are actually optional. Chunk one could contain metadata for all other chunks in the sbspace.

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 42-17](#)) 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 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 42-23](#)) in the root dbspace. For each system catalog table, Universal Server adds a one-page entry to the tblspace (see [“Structure of the Tblspace Tblspace” on page 42-19](#)) in the dbspace where the database was built. [Figure 42-46 on page 42-77](#) illustrates the relationship between the database tblspace entry and the location of the **systables** table for the database.

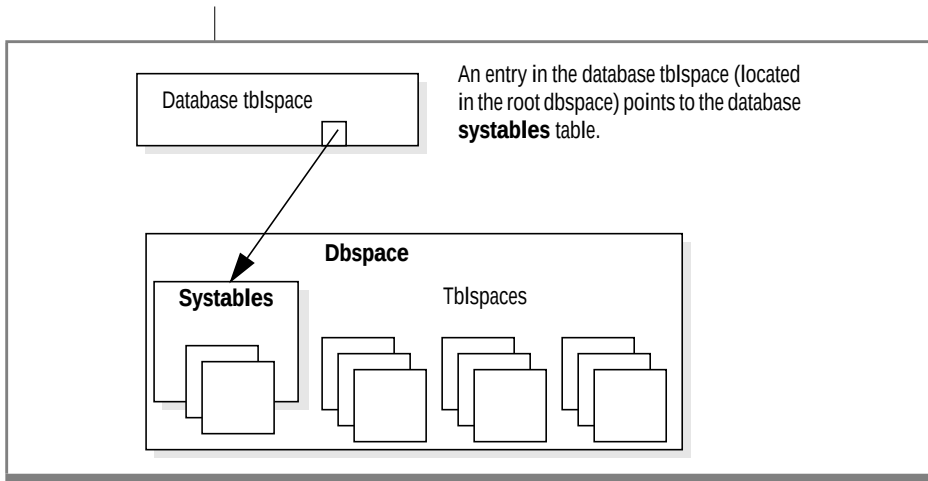


Figure 42-46
New Databases

For instructions on how to list your databases after you create them, see [“Monitoring Databases” on page 33-47](#).

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, Universal Server allocates disk space for the table in units called extents (see [“What Is an Extent?” on page 14-13](#)). The paragraphs that follow describe the major events that occur when Universal 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 42-17](#)) 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 42-20](#)) assigned to this table is derived from the logical page number in the tblspace tblspace where the table is described.

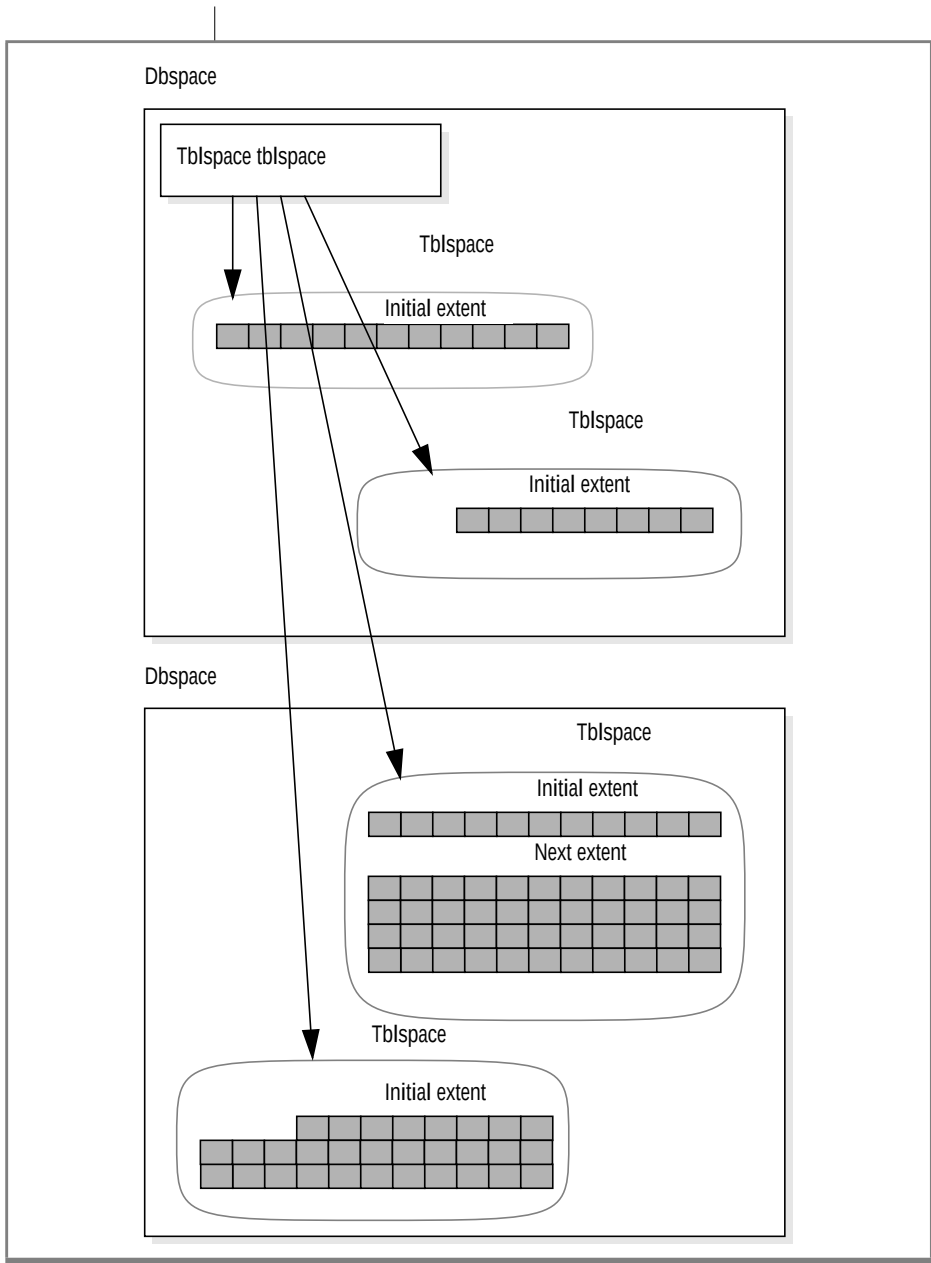
The tblspace number indicates the dbspace where the tblspace is located. Tblspace extents can be located in any of the dbspace chunks. Execute **oncheck -pe** for a listing of the dbspace layout by chunk if you must know exactly where the tblspace extents are located.

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, refer to the [Informix Guide to SQL: Reference](#).

[Figure 42-47 on page 42-79](#) illustrates the pointers within the disk data structures that track and monitor the disk space allocated to a table.

Figure 42-47
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 14-28](#).

Files That the Database Server Uses

This appendix gives brief summaries of the files that you use when you configure and use INFORMIX-Universal Server. The appendix also includes descriptions of files (and one directory) created and used internally by Universal Server and ON-Archive. For many of these files, your only responsibility is to recognize that they are legitimate.

[Figure A-1 on page A-2](#) lists the files used by INFORMIX-Universal Server and ON-Archive.

Figure A-1
List of Files Used by Universal Server and ON-Archive

Filename	Directory	Purpose	Created
af.xxx	specified by DUMPDIR configuration parameter	assertion-failure information	by Universal Server
ARCreqid.NOT	/tmp	notification file for ON-Archive	by ON-Archive
buildsmi.xxx	/tmp	error messages about SMI database	by Universal Server
config.arc	\$INFORMIXDIR/etc	configuration for archiving	during installation; modified by user informix
core	directory from which Universal Server was invoked	core dump	by Universal Server
gcore	specified by DUMPDIR configuration parameter	assertion failure information	by Universal Server
illlsrra.xx	/usr/lib	shared libraries for Universal 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
	INFORMIXTMP	create and maintain local files	by Universal Server

(1 of 3)

Filename	Directory	Purpose	Created
.infos.servername	\$INFORMIXDIR/etc	connection information	by Universal Server
.inf.servicename	/INFORMIXTMP	connection information	by Universal Server
<i>the message log;</i> filename specified by MSGPATH configuration parameter	specified by MSGPATH	error messages and status information	by Universal Server
oncatlgr.out.pidnum	/tmp	status information	by ON-Archive
the 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.serv ernum	\$INFORMIXDIR/etc	information for full-system restores	by Universal Server
oper_dflt.arc	\$INFORMIXDIR/etc	defaults for archiving	during installation; modified by user informix
servicename.exp	/INFORMIXTMP	connection information	by Universal Server
servicename.str	/INFORMIXTMP	connection information	by Universal Server

(2 of 3)

Filename	Directory	Purpose	Created
shmem.xxx	specified by DUMPDIR configuration parameter	assertion-failure information	by Universal Server
sqlhosts	\$INFORMIXDIR/etc	connection information	during installation; modified by user informix
status_vset_volnum.itgr	/tmp	error information	by ON-Archive
sysfail.pidnum	/tmp	failure information	ON-Archive
tctermcap	\$INFORMIXDIR/etc	terminal characteristics	during installation
VP.servername.nnx	/INFORMIXTMP	connection information	by Universal Server

(3 of 3)

Descriptions of Files

This section gives short descriptions of the files listed in Figure A-1.

af.xxx

Universal 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. This file is discussed in [“Monitor for Data Inconsistency” on page 31-6](#).

ARCreqid.NOT

Only the ON-Archive backup and restore system uses the file **/tmp/ARCreqid.NOT**.

The **/tmp/ARCreqid.NOT** file is a notification file. The **onarchive** utility creates this file only if the user requests a notification file *and* the **onarchive** utility is unable to write to the user's home directory. For more information, refer to the [INFORMIX-Universal Server Archive and Backup Guide](#).

buildsmi.xxx

If the database server cannot build the **sysmaster** database, it places a message in the message log that refers you to the **/tmp/buildsmi.xxx** file. This file gives information about why the build failed. For information about the sysmaster database, refer to [Chapter 38, "The sysmaster Database."](#)

config.arc

Only the ON-Archive backup and restore system uses the **\$INFORMIXDIR/etc/config.arc** file.

The **\$INFORMIXDIR/etc/config.arc** file is the ON-Archive *configuration file*. You must prepare this file if you use the ON-Archive archive and tape management system with Universal Server. The [INFORMIX-Universal Server Archive and Backup Guide](#) describes the **config.arc** file.

core

The **core** file contains a core dump caused by an assertion failure. The database server writes this file into the directory from which Universal Server was invoked. For more information, refer to ["Monitor for Data Inconsistency" on page 31-6](#).

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 DUMPPDIR configuration parameter. This file is discussed in [“Monitor for Data Inconsistency” on page 31-6](#).

illlsrra.xx

The **illlsrra.xx** files are shared libraries used by Universal Server and some Universal Server utilities. The shared libraries, if supported on your platform, are installed in **\$INFORMIXDIR/lib**. In addition, 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`

<i>lll</i>	library class (for example, “asf” or “smd”)
<i>s</i>	library subclass (d=DSA; s=standard)
<i>rr</i>	major release number (for example, “07” or “08”)
<i>a</i>	library version ID (for example, “a” or “b”)
<i>xx</i>	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. 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 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 **\$INFORMIXDIR/etc/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 Informix directory*. During initialization, Universal Server creates this directory (if it does not already exist) 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

Universal Server creates the **/INFORMIXTMP/.inf.servicename** file if any DBSERVERNAME or DBSERVERALIASES uses a shared-memory connection type. Universal Server removes the file when you take Universal Server offline. The name of this file is derived from the servicename field of the **sqlhosts** file.

The database server keeps information about client/server connections in this file. You do not use the **.inf.servicename** file 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.dbservername

Universal Server creates the **\$INFORMIXDIR/etc/.infos.dbservername** file when you initialize shared memory and removes the file when you take the database server off-line. The name of this file is derived from the DBSERVERNAME parameter in the ONCONFIG configuration file.

Universal Server uses this file to attach to utilities such as **oncheck** and **onstat**. You do not use the **.infos.dbservername** file(s) directly. You only need to recognize that the file is a legitimate file when it appears in your **\$INFORMIXDIR/etc** directory.

The Message Log

Universal Server writes status and error information into the message-log file. You can specify the filename and location of the message log with the MSGPATH configuration parameter. For more information, refer to [“MSGPATH” on page 37-46](#).

oncatlgr.out.pidnum

Only the ON-Archive backup and restore system uses the **/tmp/oncatlgr.out.pidnum** file.

The **/tmp/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**, refer to the [INFORMIX-Universal Server Archive and Backup Guide](#).

onconfig.std

The **onconfig.std** file is the *configuration-file template*. It contains default values for the ONCONFIG 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**.

[“A Sample onconfig.std File” on page A-14](#) shows a sample **onconfig.std** file. [“Overview of Configuration Parameters” on page 3-15](#) gives an overview of the parameters of the ONCONFIG configuration file. [Chapter 37, “Configuration Parameters”](#) gives a comprehensive list of the parameters in the ONCONFIG file.

onconfig

The **onconfig** file is an optional file in the **\$INFORMIXDIR/etc** directory. It contains configuration information. You can create the **onconfig** file by copying **onconfig.std** or one of your customized configuration files.

The **onconfig** file is the default configuration file. If the **ONCONFIG** environment variable is not set, the database server uses the **onconfig** file. For more information, refer to [Chapter 3, “Installing and Configuring.”](#)

The ONCONFIG File

The **\$INFORMIXDIR/etc/\$ONCONFIG** file is the *current configuration file*. This manual often refers to the current configuration file as “the ONCONFIG configuration file” or just “the ONCONFIG file.” Universal Server uses the ONCONFIG file during initialization.

To maintain compatibility with earlier versions, Universal Server also recognizes the **TBCONFIG** environment variable. However, Informix recommends that you use **ONCONFIG**.

This about
TCONFIG and
“earlier versions”
surely needs to be
changed.

If you start Universal Server with **oninit** and do not explicitly set the **ONCONFIG** environment variable, Universal Server uses the configuration file specified by the **TBCONFIG** environment variable. If neither **ONCONFIG** nor **TBCONFIG** is set, and **\$INFORMIXDIR/etc/onconfig** does not exist, Universal Server uses the **onconfig** file. If no **onconfig** file exists, Universal Server returns the following error:

```
WARNING: Cannot access configuration file $INFORMIXDIR/etc/$ONCONFIG.
```

If you start Universal Server with ON-Monitor and do not explicitly set the **ONCONFIG** environment variable, Universal Server uses the configuration file specified by the **TBCONFIG** environment variable. If neither **ONCONFIG** nor **TBCONFIG** is set, and **\$INFORMIXDIR/etc/onconfig** does not exist, ON-Monitor creates a configuration file using **onconfig.std** as a template.

For more information, refer to [Chapter 3, “Installing and Configuring.”](#)

oncfg_servername.servernum

Universal Server creates the **\$INFORMIXDIR/etc/oncfg_servername.servernum** file when you initialize disk space (**oninit -i**). The database server updates the file every time you add or delete a dbspace, a blob space, a logical-log file, or a chunk. Universal Server uses the **oncfg_servername.servernum** file 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** configuration file.

You do not use the **oncfg_servername.servernum** file(s) directly. You only need to recognize that these files are legitimate files when they appear in your **\$INFORMIXDIR/etc** directory. For more information, refer to [“Create the oncfg_servername.servernum File” on page 9-9.](#)

oper_deflt.arc

Only the ON-Archive backup and restore system uses the **oper_deflt.arc** file.

The **oper_deflt.arc** file contains default command qualifier values used by ON-Archive. The *[INFORMIX-Universal Server 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 are different from the defaults in the **oper_deflt.arc** file.

shmem.xxx

Universal Server writes information about an assertion failure into the **shmem.xxx** file. The file is stored in the directory specified by the DUMPPDIR configuration parameter. This file is discussed in [“Monitor for Data Inconsistency” on page 31-6](#).

servicename.exp

Universal Server creates an **/INFORMIXTMP/servicename.exp** file on some platforms if any DBSERVERNAME or DBSERVERALIASES uses a stream-pipe connection type. This file is used for handling expedited data, which occurs when the client program issues `sqlbreak()` or the user enters CTRL-C. The database server removes the file when you take the database server off-line. The name of this file is derived 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

Universal Server creates the **/INFORMIXTMP/servicename.str** file if any DBSERVERNAME or DBSERVERALIASES uses a stream-pipe connection type. The database server removes the file when you take the database server offline. The name of this file is derived from the servicename field of the **sqlhosts** file.

The database server keeps information about client/server connections in this file. You do not use the **.inf.servicename.str** file 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.

sqlhosts

The **\$INFORMIXDIR/etc/sqlhosts** file is the *connectivity file*. It contains information that lets an Informix client connect to an Informix database server. The **sqlhosts** file is covered in detail in [Chapter 4, “Client/Server Communications.”](#)

status_vset_volnum.itgr

Only the ON-Archive backup and restore system uses the **/tmp/status_vset_volnum.itgr** file.

The **/tmp/status_vset_volnum.itgr** file records disk-volume inconsistencies. The **onarchive** utility creates this file only when it notes an inconsistency between its catalog tables and the files located on disk. This file is described in the [INFORMIX-Universal Server Archive and Backup Guide](#).

sysfail.pidnum

Only the ON-Archive backup and restore system uses the **/tmp/sysfail.pidnum** file.

The **/tmp/sysfail.pidnum** file is created by the ON-Archive utilities when an internal fatal error occurs in **oncatlgr**, **onarchive**, or **onautovop**. You will probably never see this file. It is described in the [*INFORMIX-Universal Server Archive and Backup Guide*](#).

tctermcap

Only the ON-Archive backup and restore system uses the file **\$INFORMIXDIR/etc/tctermcap**.

The **\$INFORMIXDIR/etc/tctermcap** file defines the default user-interface attributes for the ON-Archive archive and tape-management system for the database server. The [*INFORMIX-Universal Server Archive and Backup Guide*](#) describes the **tctermcap** file.

Users can set the **ARC_KEYPAD** environment variable to point to a file that contains user-interface attributes that are different from those in the **tcterm-cap** file.

VP.servername.nnx

Universal Server creates the **/INFORMIXTMP/VP.servername.nnx** file, if needed, when you initialize shared memory. The name of this file is derived 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 the **VP.servername.nnx** file. 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 **\$INFORMIXDIR/etc/onconfig.std** file. A few 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.std
# Description: INFORMIX-Universal Server Configuration Parameters
#
*****

# Root Dbspace Configuration

ROOTNAMErootdbs      # Root dbspace name
ROOTPATH/dev/online_root # Path for device containing root dbspace
ROOTOFFSET0          # Offset of root dbspace into device (Kbytes)
ROOTSIZE20000         # 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
PHYSFILE1000         # Physical log file size (Kbytes)

# Logical Log Configuration

LOGFILES6            # Number of logical log files
LOGSIZE500           # Logical log size (Kbytes)

# Diagnostics

MSGPATH/usr/informix/online.log # System message log file path
CONSOLE/dev/console    # System console message path
ALARMPROGRAM          # Alarm program path

# System Archive Tape Device

TAPEDEV/dev/tapedev   # Tape device path
TAPEBLK16             # Tape block size (Kbytes)
TAPESIZE10240         # Maximum amount of data to put on tape
                      # (in Kbytes)
```

```
# Log Archive Tape Device

LTAPEDEV/dev/tapedev      # Log tape device path
LTAPEBLK16                # Log tape block size (Kbytes)
LTAPESIZE10240            # Max amount of data to put on log tape
                          # (in Kbytes)

# Optical

STAGEBLOB                 # INFORMIX-OnLine/Optical staging area

# System Configuration

SERVERNUM0                # Unique id corresponding to a Universal Server
                          instance
DBSERVERNAME              # Name of default database server
DBSERVERALIASES           # List of alternate dbservernames
NETTYPE                   # 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 aging
AFF_SPROC0                # Affinity start processor
AFF_NPROCS0               # Affinity number of processors

# Shared Memory Parameters

LOCKS2000                 # Maximum number of locks
BUFFERS200                # Maximum number of shared buffers
NUMAIOVPS                 # Number of asynchronous I/O VPs
PHYSBUFF32                # Physical log buffer size (Kbytes)
LOGBUFF32                 # Logical log buffer size (Kbytes)
LOGSMAX6                  # Maximum number of logical log files
CLEANERS 1                # Number of buffer cleaner processes
SHMBASE 0x0A000000L        # 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
LTXEHW60                  # Long transaction high water mark
TXTIMEOUT300              # Transaction timeout (in seconds)
STACKSIZE32               # Stack size (Kbytes)

# System Page Size

# BUFFSIZE - Universal Server no longer supports this configuration parameter.
```

```

#           To determine the page size used by Universal Server on your platform
#           see the last line of output from the command, 'onstat -b'.

# Recovery Variables
# OFF_RECVR_THREADS:
# Number of parallel worker threads during fast recovery or an offline restore.
# ON_RECVR_THREADS:
# Number of parallel worker threads during an online restore.

OFF_RECVR_THREADS  10      # Default number of offline worker threads
ON_RECVR_THREADS   1       # Default number of online worker threads

# Data Replication Variables
DRINTERVAL30       # DR max time between DR buffer flushes (secs)
DRTIMEOUT30        # DR network timeout (in sec)
DRAUTO  0          # DR automatic switchover: 0 => manual,
                  # 1 => retain type, 2 => reverse type
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:
# Universal Server equivalent of DBTEMP for SE. This is the list of dbspaces
# that the Universal Server SQL Engine will use to create temp tables etc.
# If specified it must be a colon-separated list of dbspaces that exist
# when the Universal Server system is brought online. If not specified, or if
# all dbspaces specified are invalid, various ad hoc queries will create
# temporary files in /tmp instead.

DBSPACETEMP        # 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 Universal 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
                  # Universal Server)

DUMPCNT1           # Number of shared memory or gcore dumps for
                  # a single user's session

# Index Fill Factor

FILLFACTOR90       # Fill factor for building indexes

# Method for Universal Server to use when determining current time

USEOSTIME0         # 0: use internal time (fast),

```

```
# 1: get time from OS (slow)

# Parallel Database Queries (pdq)
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

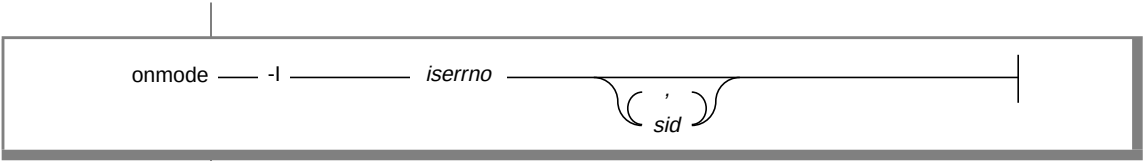
# OPTCOMPIND
# 0 => Nested loop joins are preferred (where possible) over sortmerge
#      joins and hash joins.
# 1 => If the transaction isolation mode is not "repeatable read", optimizer
#      behaves as in (2) below. Otherwise it behaves as in (0) above.
# 2 => Use costs regardless of the transaction isolation mode. Nested
#      loop joins are not necessarily preferred. Optimizer bases its
#      its decision purely on costs.
OPTCOMPIND2        # To hint the optimizer

ONDBSPACEDOWN0      # Dbspace down option: 0 = CONTINUE,
# 1 = ABORT, 2 = WAIT
LBU_PRESERVE0       # Preserve last log for log backup
OPCACHEMAX0         # Maximum optical cache size (Kbytes)
```


Trapping Errors

On occasion, a series of events causes INFORMIX-Universal Server to return unexpected error codes. If you do not have the appropriate diagnostic tools in place when these events occur, you might have difficulty determining the cause of these errors.

To assist in collecting additional diagnostics, you can use **onmode -I** to instruct Universal Server to perform the same diagnostics collection procedures described in [“Collecting Diagnostic Information” on page 31-10](#). 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 on the **onmode** utility, see [page 39-35](#).



Element	Purpose	Key Considerations
-I <i>iserrno</i>	Error number of the error for which you want to collect diagnostic information.	None.
<i>sid</i>	Session ID of the session for which you want to collect diagnostic information.	None.

Whenever Universal 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.

Index

A

- Adding listen threads 10-37
- Adding virtual processors 10-16
- ADM (Administration virtual processor) 10-16
- Administrative tasks
 - before installation 3-8
 - configuration tasks 2-5
 - consistency checking 31-3
 - controlling location of storage 14-16
 - cron jobs 3-31
 - for configuration 3-13
 - in quiescent mode 7-4
 - initial tasks 2-3
 - list of tasks 3-29
 - list of tools 3-6
 - managing the sqlhosts file 4-19
 - of database administrator 1-17
 - of Universal Server administrator 1-18
 - planning 3-4
 - routine tasks 2-4
 - situations to avoid 32-3
 - startup and shutdown scripts 3-29
 - types of 2-3
- ADT (audit) virtual processor 10-39
- ADTMODE parameter
 - mentioned 10-39
- AFF_NPROCS parameter
 - and VPCLASS 37-7
 - description of 37-7
 - purpose of 10-22
- AFF_SPROC parameter
 - and VPCLASS 37-8
 - description of 37-8
 - processor affinity 10-23
 - purpose of 10-22
- af.xxx file A-4
- Aggregate, parallel processing
 - of 10-10
- AIO virtual processors
 - how many 10-30
 - NUMAIOVPS parameter 10-29
 - when used 10-29
- ALARMPROGRAM
 - parameter 33-8
- ALL keyword
 - with DATASKIP 16-23
- Allocating disk space
 - extent 14-14
 - for mirrored data 28-5
 - initial configuration 3-26
 - procedure 15-4
- ALTER TABLE statement
 - mentioned 4-45
- Alternate dbservername 37-14
- ANSI compliance
 - level Intro-14
- ANSI-compliant transaction
 - logging. *See* Logging.
- Application
 - client. *See* Client application.
 - developer, role of 1-18
- Application developer
 - role in managing PDQ 19-9
- Applications, types of 18-6
- Architecture
 - central principles 10-3
 - description of 10-3

- Archive
 - and fragmentation 16-6, 16-28
 - preparing environment in
 - multiple-residency setting 6-8
 - reserved page information 42-14
 - tools for 3-19
 - using ontape 39-107
 - with multiple residency 6-8
- Archive configuration file A-5, A-11
- Archive interface attributes
 - file A-13
- Archiving
 - strategy 3-4
- ARCrequid.NOT file,
 - description A-5
- ARC_DEFAULT environment
 - variable
 - setting with informix.rc 3-32
- ARC_KEYPAD environment
 - variable A-11, A-13
- arp command 4-37
- ASF 4-10
- ASF. *See* Connectivity.
- Assertion failure
 - and data corruption 31-9
 - description of 31-3
 - determining cause of 31-11
 - DUMPCNT parameter 37-27
 - DUMPCORE parameter 37-27
 - DUMPSHMEM parameter 37-29
 - during consistency checking 31-6
 - during processing of user
 - request 31-8
 - form in message log 31-6
- Assertion failure file A-4
 - af.xxx 31-7
 - gcore A-6
 - gcore.xxx 31-7
 - list of 31-7
 - shmem.xxx 31-7, A-11
- Asterisk, as wildcard in hostname
 - field 4-32
- Asynchronous I/O 10-28
 - and NUMAIOVPS
 - parameter 37-50
 - write requests for mirrored
 - data 27-9
- Attached index 16-17

- Attaching to shared memory
 - additional segments 12-15
 - client to communications
 - portion 12-11
 - description of 12-11
 - Universal Server utilities 12-12
 - virtual processors 12-12
 - virtual processors and key
 - value 12-13
- Audit mode 10-39
- Audit records
 - and sysaudit table 38-10
- Auditing 1-16, 38-9, 38-10
- Authentication
 - default policy 4-7
 - description of 4-7
- Automatic recovery, by two-phase
 - commit protocol 34-10
- Availability
 - and critical media 14-16
 - and data replication 29-4
 - as goal in efficient disk
 - layout 14-36
 - improving with
 - fragmentation 16-5
 - sample disk layout 14-43

B

- Backup
 - active users, and 1-14
 - dbspace backup, mentioned 1-14
 - displaying contents of 39-30
 - freeing a log file 23-14
 - hot site 29-4
 - mentioned 14-35
 - of Simple large objects 14-23
 - planning for 3-5
 - reducing size of 14-22
 - strategy 3-4
 - transaction records 1-14
 - with multiple residency 6-8
- Bad-sector mapping, absence
 - of 1-19
- Before-image
 - contents 26-6
 - described 24-3
 - flushing of 12-46
 - in physical log buffer 12-46
 - journal. *See* Physical log.
 - role in buffer modification 12-45
- Big buffers
 - and regular buffers 12-30
 - description of 12-29
- Big-remainder page 42-41
- Binary large object
 - mentioned 1-7
- Binding CPU virtual processors
 - and AFF_NPROCS
 - parameter 37-7
 - benefit of 10-11
 - parameters 10-23
- Bit-map page
 - 2-bit values describing
 - fullness 42-25
 - 4-bit values describing
 - fullness 42-26
 - component of a tbspace 14-31
 - component of an extent 42-28
 - component of the tbspace
 - tbspace 42-21, 42-22
 - description of 42-24
 - location in extent 42-24
 - of a blobspace 42-67
 - role of 4-bit page 42-26
 - structure of 42-24
 - types of entries 42-24
- BLOB data type
 - mentioned 1-7
- Blobpage
 - average fullness statistics 39-12, 39-21
 - blobpage size 42-62
 - blobpage size and storage
 - efficiency 15-21
 - components of page header 42-69
 - description of 14-11
 - freeing deleted pages 22-24
 - fullness terminology
 - explained 15-24
 - interpreting average
 - fullness 15-24
 - oncheck -pB display
 - explained 15-22
 - relationship to chunk 14-11
 - size of 14-11
 - sizing for performance 15-21

- sizing recommendations 14-12
- specifying size of 42-62
- storage statistics 15-22
- structure and storage 42-62, 42-68
- writes bypass shared memory 12-61
- Blobspace
 - activating 22-23
 - adding a chunk 15-12, 15-16
 - bit-map page 42-67
 - blobpage structure 42-68
 - creating 3-28
 - creating using ON-Monitor 15-15
 - creating using onspaces 15-16
 - description of 14-23
 - dropping using ON-Monitor 15-20
 - dropping using onspaces 15-20
 - dropping, initial tasks 15-19
 - free-map page
 - description of 42-67
 - location in blobspace 42-63
 - role in blobpage logging 22-25, 42-67
 - role in tracking blobpages 12-63
 - tracked by bit-map 42-67
 - illustration of writing a simple large object 12-63
 - logical-log administration tasks for 22-23
 - multiple residency 6-7
 - overhead pages 22-24
 - page types 42-67
 - purpose of 14-23
 - restriction concerning dropping 39-63
 - Simple-large-object buffers 12-62
 - Simple-large-object storage 42-64
 - Simple-large-object timestamps 12-59
 - storage efficiency 15-21
 - storage statistics 15-22
 - structure 42-62
 - structure of blobspace mirror
 - chunk 42-16
 - writing data to 12-61
- BLOB. *See* Binary large object.
- Block device 14-6
- Boot file. *See* Startup script.

- B-tree
 - cleaner list 42-57
 - functional index 42-60
 - indexing 42-48
 - structure 42-48
- Buffer
 - access-level flag bits 39-84
 - big buffers 12-29
 - blobpage buffer 12-62
 - concurrent access 12-42
 - current lock-access level for 12-20
 - data replication 12-28, 29-10
 - dirty 12-45
 - exclusive mode 12-36
 - flushing 12-45
 - how a thread accesses a buffer page 12-42
 - how a user thread acquires 12-38
 - how thread releases after modification 12-45
 - least-recently used 12-37
 - lock types 12-35
 - lock-access level of 12-43
 - logical-log buffer 12-26, 12-45
 - maximum number 12-24
 - minimum requirement 12-24
 - monitoring statistics and use of 33-21
 - most-recently used 12-37
 - not dirty 12-45
 - page-type codes 39-83
 - physical-log buffer 12-27, 12-45
 - reading from disk 12-43
 - regular buffer 12-24
 - releasing if no thread waiting 12-44
 - releasing if thread waiting 12-44
 - releasing when modified 12-44
 - releasing when not modified 12-44
 - share lock 12-35
 - status 12-20
 - synchronizing flushing 12-45
 - threads waiting for 12-20
 - what occurs when modified 12-44
 - write types during flushing 12-50

- Buffer flushing
 - description of 12-45
 - how synchronized 12-49
- Buffer pool
 - bypassed by blobspace data 12-61
 - contents of 12-24
 - description of 12-24
 - flushing 12-46
 - LRU queues management 12-37
 - monitoring activity 33-24
 - read-ahead 12-41
 - synchronizing buffer flushing 12-49
- Buffer table
 - contents of 12-20
 - description of 12-20
 - LRU queues 12-36
- Buffered logging, when flushed 20-11
- Buffered transaction logging. *See* Logging.
- Buffers
 - smart large objects, and 12-64
- BUFFERS parameter
 - description of 12-24, 37-9
 - mentioned 3-21
 - smart large objects, and 12-64
- Buffer-size option, in options field 4-42
- BUFFSIZE. *See* Page size.
- buildsmi script
 - buildsmi.xxx file, description A-5
 - error log message 40-16
 - failure of A-5
 - when initializing Universal Server 38-4
- Built-in data types 1-4
 - description of 1-6
- BYTE data type
 - Committed Read isolation 12-61
 - Dirty Read isolation 12-60
 - mentioned 1-7
 - requires 4-bit bit map 42-26, 42-28
 - storage on disk 14-18
- Byte lock 12-22
- B+ tree. *See* B-tree.

C

Cache. *See* Shared-memory buffer pool.

Cascading deletes 20-5

Casting

user-defined, mentioned 1-4

Cataloger, ON-Archive,
mentioned 6-8

Cautions 32-3

Character large object
mentioned 1-7

Character-special interface 14-6

Checkpoint

and chunk writes 12-51

and flushing of regular
buffers 12-46

and logical-log file 12-56

and physical-log buffer 12-56,
24-11

description of 12-56

events that initiate 12-56

force with onmode -c 39-39

forcing, to free logical-log
file 13-16

mentioned 9-9

monitoring activity 33-17

reserved page information 42-8

role in data replication 29-14

role in fast recovery 12-58, 26-5,
26-6

step in shared-memory

initialization 9-9

updating the reserved pages 42-7

Chunk

activity during mirror
recovery 27-8

adding to a blob space 15-16

adding to a db space 15-12

adding to a sb space 15-12

adding to mirrored db space 28-9

adding using ON-Monitor 15-13

adding using onspaces 15-13

and associated partitions 14-37

changing mirror chunk
status 28-10

checking for overlap 39-20

chunk status as PD 31-10

cooked versus raw 14-6

creating a link to the
pathname 15-7, 28-5

defining multiple in a
partition 15-6

description of 14-5

dropping from a blob space or
sb space 15-19

dropping from a db space 15-18

exceeding size limits with

LVM 14-49

free-list page 42-15, 42-16, 42-17,
42-18

general disk layout

guidelines 14-37

initial chunk of db space 42-4

initial mirror offset 37-45

I/O errors during

processing 31-10

limits on size and number 14-6

maximum number of 14-6

maximum size of 14-6

mirror chunk reserved page

information 42-13

monitoring 33-55, 33-60, 38-11

multiple residency 6-7

name, when allocated as raw
device 14-6

pathname stored 42-11

purpose of 14-5

recovering a down chunk 28-10

relation to extent 14-14

relinking after disk failure 28-12

reserved page information 42-11

structure

additional db space chunk 42-16

initial db space chunk 42-15

mirror chunk 42-16

using a symbolic link for the
pathname 37-61

Chunk free list

how checked by oncheck 39-11,
39-13, 39-19

Chunk table

and mirroring 12-20

contents of 12-21

description of 12-20

Chunk write

checkpoints 12-51

monitoring 33-25

CKPTINTVL parameter

description of 37-11

initiating checkpoints 12-56

mentioned 3-22

Classes of virtual processor 10-7

CLEANERS parameter

description of 37-12

purpose of 12-22

Client

and USEOSTIME configuration
parameter 37-74

client id 39-89

connecting to primary server 29-5

connecting to secondary

server 29-5

description of 1-10

designing for redirection 29-22

enabling communication 4-18

finding and connecting to a
host 4-24

improving performance and
availability 29-4

killing a client session 39-40

loopback connection 4-14

network connection 4-13

printing client sessions 39-88

reacting to failure 29-17

redirecting and connectivity in
data replication 29-22

redirecting in data

replication 29-17

remote 4-21

results of connection 37-51

security requirements 4-21

shared-memory connection 4-11

specifying a dbservername 4-44

specifying dbservername 37-15

stream-pipe connection 4-11

Client application

connecting to Universal

Server 4-25

redirecting in data

replication 29-22

testing 5-4

wildcard addressing 4-34

Client/server architecture

client/server connection 1-10

description of 1-10

- Client/server configuration
 - connections 4-10
 - example
 - 7.2 client with 5.x server 4-58
 - 7.2 relay module 4-53
 - INFORMIX-STAR,
 - example 4-57
 - local loopback 4-48
 - multiple connection types 4-50
 - multiple residency 4-52
 - multiple Universal Server
 - servers 4-52
 - network connection 4-49
 - shared memory 4-47
 - using IPX/SPX 4-50
 - listen and poll threads 10-33
 - local loopback 4-14
 - remote host 4-13
 - shared memory 4-11
- CLOB data type
 - see Character large object 1-7
- CLOB. *See* Character large object.
- Cold restore, number of recovery
 - threads 37-52
- Command-line conventions
 - elements of Intro-9
 - example diagram Intro-11
 - how to read Intro-11
- Comment icons Intro-8
- COMMIT statement, unbuffered
 - logging 20-10
- Committed Read isolation level
 - data-consistency checks 12-61
 - role of simple-large-object
 - timestamps 12-60
- Commit, heterogeneous 34-38
- Communication configuration file.
 - See* ONCONFIG configuration file.
- Communication support services
 - description of 4-6
- Communications portion of shared
 - memory
 - how client attaches 12-11
- Communications portion (shared
 - memory)
 - contents of 12-33
 - description of 12-33
 - size of 12-33

- Communications support module
 - configuring 4-8
 - description of 4-7
 - sqlhosts file 4-8
 - sqlhosts option field 4-43
- Communication, client to database
 - server. *See* Connectivity.
- Compactness, of index page 37-30
- Complex data types 1-4
 - description of 1-6
- Compliance, with industry
 - standards Intro-14
- concsn.cfg file
 - configuring the communications
 - support module 4-8
 - format of entries 4-23
 - location of 4-9
 - specifying the communications
 - support module 4-9
- Concurrency control 12-34
- Concurrency, improving with
 - fragmentation 16-5
- Concurrent scans
 - limiting the number 19-8
- Configuration
 - backups 3-19
 - database server
 - identification 3-17
 - disk space 3-16
 - error dumps 3-26
 - estimating required disk
 - space 14-36
 - fragmentation 3-23
 - logical log 3-18
 - logical recovery 3-25
 - message logs 3-20
 - mirroring 3-17
 - monitoring 33-15
 - optical storage 3-26
 - overview 3-10
 - parameter overview 3-15
 - PDQ 3-23
 - physical log 3-19
 - planning for Universal Server 3-4
 - preparing the environment 3-13
 - root dbspace 3-15
 - shared memory 3-21
 - shared-memory allocation 3-22
 - shared-memory buffer pool 3-22

- template file 3-11
- time intervals 3-25
- using 7.2 relay module 4-54
- virtual processors 3-24
- Configuration file
 - and multiple residency 5-3, 6-4
 - archive A-5
 - connectivity 4-18
 - onconfig.std 3-11
 - reserved page information 42-8
 - warning about multiple
 - residency 6-4
- Configuration parameter
 - See also* Configuration parameter use.
 - See also each parameter listed under its own name.*
 - AFF_NPROCS 37-7
 - AFF_SPROC 37-8
 - ALARMPROGRAM 33-8
 - BUFFERS 37-9
 - CKPTINTVL 37-11
 - CLEANERS 37-12
 - CONSOLE 3-15, 37-12
 - DATASKIP 16-21, 17-4
 - DBSERVERALIASES 4-45, 37-14
 - DBSERVERNAME 3-14, 4-27, 4-44, 37-15
 - DBSPACETEMP 37-16
 - DEADLOCK_TIMEOUT 37-18
 - DRAUTO 37-18
 - DRINTERVAL 37-19
 - DRLOSTFOUND 37-20
 - DRTIMEOUT 37-20
 - DS_MAX_QUERIES 18-13, 37-21
 - DS_MAX_SCANS 18-13, 37-22
 - DS_TOTAL_MEMORY 18-14, 37-23
 - DUMPCNT 37-27
 - DUMPCORE 37-27
 - DUMPDIR 37-28
 - DUMPGCORE 37-28
 - DUMPSHMEM 37-29
 - FILLFACTOR 37-30
 - HETERO_COMMIT 37-30
 - LOCKS 37-32
 - LOGBUFF 37-33
 - LOGFILES 37-34
 - LOGSIZE 37-35

- LOGSMAX 37-36
- LRUS 12-38, 37-37
- LRU_MAX_DIRTY 37-37
- LRU_MIN_DIRTY 37-38
- LTAPEDEV 3-15
- LTXEHWM 37-41
- LTXHWM 37-42
- MAX_PDQPRIORITY 18-13, 37-43
- MIRROR 37-44
- MIRROROFFSET 37-45
- MIRRORPATH 37-45
- MSGPATH 3-15, 37-46
- MULTIPROCESSOR 37-46
- NETTYPE 4-44, 37-47
- NOAGE 37-49
- NUMAIOVPS 37-50
- NUMCPUVPS 37-51
- OFF_RECVRY_THREADS 37-52
- ON_RECVRY_THREADS 37-52
- OPTCOMPIND 18-14, 19-10, 37-54
- PHYSBUFF 37-56
- PHYSDBS 37-57
- PHYSFILE 37-57
- RA_PAGES 37-58
- RA_THRESHOLD 37-59
- RESIDENT 37-60
- ROOTNAME 3-15
- ROOTOFFSET 37-61
- ROOTPATH 3-14, 37-60, 37-61
- ROOTSIZE 3-16, 37-62
- SBS spacename 37-62
- SERVERNUM 3-14, 12-12, 37-63
- SHMBASE 12-12, 37-65
- SHMVIRTsize 37-67
- SINGLE_CPU_VP 37-68
- STACKSIZE 37-69
- STAGEBLOB 37-70
- TAPEDEV 3-15
- TXTIMEOUT 37-73
- USEOSTIME 37-74
- VPCLASS 37-75
- Configuration parameter use
 - and initial chunk of root
 - dbspace 14-21
 - displayed
 - in data-replication screen 36-15
 - in diagnostics screen 36-16
 - in initialization screen 36-12
 - in PDQ screen 36-17
 - in performance screen 36-14
 - in shared-memory screen 36-13
 - enabling Logs Full HWM 22-10
 - for archiving 3-19
 - for diagnostic information 31-10
 - for multiple residency 5-4
 - for ontape utility 3-20
 - for physical logging 3-19
 - identification 3-17
 - shared memory 3-21, 13-3
- Configure
 - extspaces 3-29
- config.arc file
 - description of A-5
 - with multiple residency 6-8
- CONNECT statement
 - example 4-45
 - mentioned 4-25, 4-45
- Connecting
 - and multiple residency 6-6
 - description of 10-33
 - example with CONNECT 4-25
 - methods 10-31
 - to non-Informix databases 1-16
 - with sockets 10-31
 - with TLI 10-31
- Connection
 - 7.2 client with 5.x server,
 - example 4-58
 - IPX/SPX 4-50
 - local loopback, definition of 4-14
 - local loopback, example 4-48
 - multiple connection types
 - example 4-50
 - multiple residency, example 4-52
 - network, description of 4-13
 - network, example 4-49
 - network, when to use 4-13
 - security restrictions 4-21
 - shared memory, description
 - of 4-11
 - TCP/IP 4-21
 - to client application 4-10
 - to database server 4-10
- Connectivity
 - configuration file 4-18
 - configuration parameters 4-44
 - file, sqlhosts 3-11
- Consistency checking
 - corruption of data 31-8
 - data and overhead 31-4
 - index corruption 31-9
 - monitoring for data
 - inconsistency 31-6
 - overview 31-3
 - periodic tasks 31-3
 - validating extents 31-5
 - validating indexes 31-5, 31-6
 - validating reserved pages 31-4
 - when to schedule 31-4
- Console messages 33-12
- CONSOLE parameter
 - changing 33-12
 - description of 37-12
 - mentioned 3-20
 - setting 3-15
- Constraint, deferred checking 20-5
- Contention. *See* Disk contention.
- Context switching
 - description of 10-13
 - how functions when OS
 - controls 10-9
 - OS versus multithreaded 10-9
- Contiguous space for physical
 - log 25-4
- Control structures
 - description of 10-12
 - queues 10-16
 - session control block 10-12
 - stacks 10-15
 - thread control block 10-13
- Conversion
 - during initialization 9-8
- Cooked file space
 - and buffering 14-8
 - compared with raw space 14-6, 14-7
 - contiguity of space 14-8
 - description of 14-6
 - for static data 14-8
 - how to allocate 15-5
 - in data storage 14-3
 - rationale for using 14-8

- reliability 14-8
- Universal Server management
 - of 14-7
 - warning 14-7
- Coordinating database server
 - and automatic recovery 34-11
 - description of 34-5
- Coordinator recovery
 - mechanism 34-11
- Core dump
 - and DUMPCORE configuration
 - parameter 37-27
 - contained in core file A-5
 - contents of gcore.xxx 31-7
 - when useful 31-11
 - See also* DUMPCNT, DUMPDIR, DUMPGCORE, DUMPSHMEM.
- core.pid.cnt file 37-29
- Correlated subquery, effect of
 - PDQ 18-19
- Corruption
 - corrective actions 31-9
 - determining if exists 31-9
 - I/O errors from a chunk 31-10
 - symptoms of 31-9
- CPU
 - binding to virtual processor 10-11
 - relationship to virtual
 - processor 10-6
 - time tabulated 39-98
- CPU virtual processor
 - adding and dropping in on-line
 - mode 10-21, 10-26
 - AFF_NPROCS parameter 10-22
 - AFF_SPROC parameter 10-22
 - and poll threads 10-32, 10-33
 - and SINGLE_CPU_VP
 - parameter 37-68
 - binding 37-7
 - DataBlade module on 10-19
 - description of 10-19
 - how many 10-19
 - on a multiprocessor
 - computer 10-20
 - on a single-processor
 - computer 10-20, 10-21
 - preventing priority aging 10-21
 - restrictions on 10-24

- the NUMCPUVPS
 - parameter 10-19
 - types of threads run by 10-19
 - user-defined routine on 10-19
- CREATE FUNCTION
 - statement 10-25
 - and VPCLASS 37-78
- CREATE INDEX statement
 - effect of PDQ 18-17
 - using FILLFACTOR 37-30
- CREATE TABLE statement
 - mentioned 4-45
- Critical media
 - mirroring 14-37, 14-47
 - storage of 14-16
- Critical section of code
 - and checkpoints 12-57
 - and filling of physical-log
 - buffer 12-48
 - description of 12-55
 - related to size of physical log 24-5
- cron jobs, warning about 3-31

D

- Data block. *See* Page.
- Data consistency
 - fast recovery 26-3
 - how achieved 12-55
 - monitoring for 31-6
 - symptoms of corruption 31-9
 - timestamps 12-59
 - verifying consistency 31-4
- Data definition statements, when
 - logged 20-6
- Data files. *See* Logging.
- Data management 14-7
- Data manipulation statements,
 - when logged 20-7
- Data pages
 - and oncheck 39-11, 39-17
- Data replication
 - actions to take if primary
 - fails 29-18
 - actions to take if secondary
 - fails 29-17
 - administration of 30-14
 - advantages of 29-4
- and the PAGE_ARCH reserved
 - page 42-14
- automatic switchover 29-18
- changing database server
 - mode 30-19
- changing database server
 - type 30-20
- client redirection
 - comparison of different
 - methods 29-32
 - handling within an
 - application 29-29
 - using DBPATH 29-23
 - using
 - INFORMIXSERVER 29-28
 - using sqlhosts file 29-24
- configuring connectivity for 30-8
- database and data requirements
 - for 30-5
- database server configuration
 - requirements for 30-6
- description of 1-15, 29-3
- designing clients to use secondary
 - database server 29-34
- designing clients to use the
 - primary server 29-33
- detecting failures of 29-16
- DRAUTO parameter 29-18
- DRINTERVAL parameter 29-11
- DRLOSTFOUND
 - parameter 29-12
- DRTIMEOUT parameter 29-16
- flush interval 37-19
- hardware and operating system
 - requirements for 30-4
- how it works 29-8
- how updates are replicated 29-9
- importance of reliable
 - network 29-20
- information in sysdri table 38-17
- initial replication 29-8
- lost-and-found file 37-20
- lost-and-found transactions 29-11
- manual switchover 29-21
- mentioned 20-5
- monitoring status 33-76
- planning for 30-3
- possible failures 29-15
- read-only mode 7-4

- restarting after failure 29-19, 29-21, 30-26
- restarting after network failure 30-29
- restoring system after media failure 30-22
- role of checkpoint 29-14
- role of log records 29-9
- role of primary database server 29-4
- role of secondary database server 29-4
- role of temporary dbspaces 29-36
- setting up 30-4
- specialized threads 29-13
- starting 30-10
- synchronization 29-14
- wait time for response 37-20
- with asynchronous updating 29-11
- with synchronous updating 29-11
- Data row
 - and rowid 42-38
 - big-remainder page 42-41
 - forward pointer 42-39, 42-42
 - home page 42-37, **42-41**
 - how Universal Server locates 42-38
 - Simple large object descriptor component 42-64
 - storage strategies 42-37
 - storing data on a page 42-40
- Data storage
 - control of 14-17
 - limits on due to maximum chunk size 14-6
 - overview 14-3
 - types of 14-3
 - See also* Disk space.
- Data type
 - creating new 2-6
- Data Type segment
 - See also* Disk space.
- Data types
 - BLOB, mentioned 1-7
 - built-in, description of 1-6
 - built-in, mentioned 1-4
 - BYTE, mentioned 1-7
 - CLOB, mentioned 1-7
 - complex
 - mentioned 1-4
 - complex, description of 1-6
 - opaque, mentioned 1-6
 - TEXT, mentioned 1-7
 - user-defined, description of 1-6
 - user-defined, mentioned 1-4
- Database
 - controlling storage location 14-27
 - creating, what happens on disk 42-76
 - description of 14-25
 - effects of creation 42-76
 - estimating disk space for 14-36
 - estimating size of 14-36
 - information in sysdatabases table 38-14
 - locale, in sysdbsllocale table 38-15
 - location of 14-25
 - migration. *See* Migration.
 - monitoring 33-47
 - owner, in sysmaster
 - database 38-14
 - purpose of 14-25
 - recovery. *See* Recovery.
 - size limits 14-26
 - tuning. *See* Performance tuning.
- Database administrator
 - See also* Administrative tasks.
- Database I/O 10-28
- Database logging status
 - ANSI-compliant,
 - description 20-11
 - buffered, description 20-11
 - changes permitted 21-3
 - changes, general info 21-4
 - changing buffering status
 - using ON-Archive 21-6
 - using ontape 21-8
 - description of 20-10
 - ending logging
 - using ON-Archive 21-6
 - using ontape 21-8
 - in a distributed
 - environment 20-12
 - making ANSI-compliant
 - using ON-Archive 21-7
 - using ontape 21-8
 - modifying
 - using ON-Archive 21-5
 - using ON-Monitor 21-9
 - using ontape 21-7
 - setting 20-9
 - turning on logging
 - using ON-Archive 21-5
 - using ontape 21-7
 - unbuffered, description 20-10
 - who can change 20-13
- Database schema. *See* Informix Migration Guide.
- Database server
 - connection to 4-10
 - description of 1-3
 - remote 4-22, 38-27
- Database server environment
 - configuration 3-13
- Database server name. *See* dbservername.
- DATABASE statement,
 - mentioned 4-45
- Database tblspace
 - entries 42-23
 - location in root dbspace 42-4, 42-23
 - relation to systable 42-76
 - structure and function 42-23
 - tblspace number 42-23
- DataBlade modules
 - description of 1-8
 - installing 3-6
 - mentioned 1-4
 - virtual processors for 10-19
- Data-recovery mechanisms
 - fast recovery 26-3
 - mirroring 27-4
- Data-replication buffer 12-28, 29-9
- DATASKIP parameter 16-21
 - specifying with onspaces utility 39-74
 - using 16-21, 17-4
 - with ALL keyword 16-23
- Data, estimating disk space
 - for 14-34
- DB-Access
 - testing configuration 4-58
- dbaccessdemo7 script Intro-5
- DBNETTYPE environment
 - variable 4-55

- DBPATH environment variable
 - use in automatic redirection 29-23
- DBSERVERALIAS parameter
 - and multiple residency 5-4
 - description of 4-45, 37-14
 - example 4-45
 - in sqlhosts file 4-27
 - mentioned 3-17
 - multiple connection types
 - example 4-50
- dbservername
 - conflict with INFORMIX-SE 37-15
 - description of 37-15
 - field in sqlhosts file 4-27
 - in CONNECT statement 4-25
 - mentioned 1-11
 - purpose of 4-27
 - syntax rules 4-26
- DBSERVERNAME parameter
 - and multiple residency 5-4
 - associated protocol 10-32
 - description of 4-44, 37-15
 - in sqlhosts file 4-27
 - mentioned 3-17
 - multiple residency 6-5
 - setting 3-14
 - virtual processor for poll thread 10-32
- Dbospace
 - adding a chunk to 15-12
 - adding a mirrored chunk 28-9
 - as link between logical and physical units of storage 14-17
 - bit-map page 42-24
 - creating a temporary 15-10
 - creating during initial configuration 3-28
 - creating with ON-Monitor 15-10
 - creating with onspaces 15-11
 - creating, overview 15-9
 - description of 14-16
 - dropping a chunk from 15-18
 - dropping using ON-Monitor 15-20
 - dropping using onspaces 15-20
 - dropping, overview 15-19

- identifying the dbspace for a table 42-21
- initial dbspace 14-21
- list of structures contained in 42-15
- mirror chunk information 42-7
- mirroring if logical log files included 27-7
- modifying with onspaces utility 39-74
- monitoring simple large objects 33-74
- monitoring with SMI 38-15
- multiple residency 6-7
- page header 42-34
- primary chunk information 42-7
- purpose of 14-16
- reserved page information 42-11
- role in fragmentation 16-3
- root dbspace defined 14-21
- root name 37-60
- shared-memory table 12-21
- Simple-large-object page structure 42-65
- simple-large-object storage 42-64
- simple-large-object
 - timestamps 12-59
- starting to mirror 28-8
- storage 42-4
- structure 42-4, 42-15, 42-16
- structure of additional dbspace chunk 42-16
- structure of chunk free-list page 42-17
- structure of mirror chunk 42-16
- structure of nonroot dbspace 42-15
- structure of tblspace
 - tblspace 42-19
- temporary 14-21
- usage report 42-5
- Dbospace table
 - contents of 12-21
 - description of 12-21
- DBSPACETEMP environment variable 14-30
- DBSPACETEMP parameter
 - and load balancing 14-38
 - description of 37-16

- if not set 14-30
- mentioned 3-16
- relationship to DBSPACETEMP environment variable 14-29
- DCE components
 - installing and configuring 4-8
- DCE-GSS communications support module
 - configuring 4-8
 - description of 4-7
 - shared library 4-7
 - sqlhosts 4-8
- Deadlock prevention 22-8
- Deadlock. *See* DEADLOCK_TIMEOUT parameter.
- DEADLOCK_TIMEOUT parameter
 - description of 37-18
 - in two-phase commit 34-37
 - mentioned 3-25
- Decision-support application and fragmentation 16-25
 - characteristics 18-8
 - compared to OLTP 18-7
 - definition of 18-7
 - uses of 18-7
- Decision-support query
 - DS_MAX_QUERIES
 - configuration parameter 18-13
 - DS_MAX_QUERIES parameter 37-21
 - DS_TOTAL_MEMORY parameter 37-23
 - MAX_PDQPRIORITY parameter 37-43
 - monitoring resources allocated for 33-37, 33-40
 - monitoring threads for 33-36, 33-44
 - parameters, setting with onmode 39-46
 - use of
 - DS_TOTAL_MEMORY 18-14
- See also* PDQ.
- Default configuration file 3-12, 9-6, A-9
- DEFAULT keyword, with SET DATASKIP 17-6

- Default locale Intro-5
- Deferred checking of
 - constraints 20-5
- Delete flag
 - described 42-49
- Demonstration database Intro-5
- Detached index 16-18
- Device 14-6
- Diagnostic information
 - and disk space restraints 31-11
 - collecting 31-10
 - parameters to set 31-10
- Diagnostic messages. *See* Message log.
- Diagnostics
 - using onmode B-1
- Dictionary cache 12-31
- Dirty buffer, description of 12-45
- Dirty Read isolation level
 - data-consistency checks 12-60
 - role of simple large object timestamps 12-60
- Disabling I/O error
 - circumstances under which they occur 14-15
 - defined 14-14
 - destructive versus nondestructive 14-14
 - monitoring using event alarms 33-59
 - monitoring using message log 33-58
- Disk contention
 - and high-use tables 14-40
 - of critical media 14-38
 - reducing 14-36, 16-26
- Disk failure 1-14
- Disk I/O
 - kernel asynchronous I/O 10-26
 - logical log 10-26
 - managing 14-3
 - operating system I/O 14-6
 - physical log 10-26
 - priorities 10-27
 - raw I/O 14-6
 - reads from mirrored chunks 27-9
 - role of shared memory in reducing 12-5
 - virtual processor classes 10-26
 - writes to mirrored chunks 27-9
- Disk I/O queues 10-30
- Disk layout
 - and archiving 14-39, 14-41
 - and logical volume managers 14-49
 - and mirroring 14-41
 - and table isolation 14-40
 - for optimum performance **14-37**
 - sample disk layouts 14-43
 - trade-offs 14-43
- Disk page
 - before-images in physical log 12-46
 - function of timestamp pairs 12-59
 - logical page number 42-38
 - number to read ahead 37-58
 - page compression 42-36, **42-46**
 - physical page number 42-38
 - read ahead 12-41
 - storing data on a page 42-40
 - structure
 - blobspace blobpage 42-29
 - dbspace page 42-34
 - types of pages in an extent 42-28
- Disk space
 - allocating
 - raw disk space 14-8
 - when a database is created 42-76
 - when a table is created 42-77
 - allocating cooked file space 15-5
 - allocating raw disk space 15-5
 - allocation for system catalogs 42-76
 - caution with multiple residency 6-7
 - chunk free-list page 42-17
 - configuring for multiple residency 5-6
 - creating a link to chunk pathname 15-7
 - described 15-4
 - efficient storage of simple large objects 15-22
 - estimating size of 14-33
 - initialization 9-8
 - definition of 9-3, 15-8
 - with new database server 3-28
 - with oninit 15-9, 39-27
 - with ON-Monitor 15-8
 - layout guidelines 14-36
 - limits on due to maximum chunk size 14-6
 - list of structures 42-3
 - locating free space 42-18
 - multiple residency 6-7
 - offsets for chunk pathnames 15-6
 - optimal storage of tables 14-42
 - optimizing temporary space layout 14-38
 - page compression 42-36, **42-46**
 - raw devices versus cooked files 14-6
 - requirements 14-36
 - for root dbspace 14-33
 - strategies for improving performance 14-36
 - temporary. *See* Temporary disk space.
 - tracking
 - available space in a blobspace 42-67
 - available space in a chunk 42-17
 - free pages with bit-map page 42-24
 - usage by tblspace 14-31
- Disk volume inconsistency
 - information A-12
- Distributed databases 1-16
- Distributed environment, logging status 20-12
- Distributed queries, description of 1-16
- Distributed transaction
 - and two-phase commit protocol 34-4
 - determining if inconsistently implemented 35-6
 - mentioned 20-5
- Distribution scheme
 - choosing 16-29
 - description of 16-3
 - expression-based classes of 16-8
 - with arbitrary rule 16-11

- with hash rule 16-11
- with range rule 16-10
- round robin 16-7, 16-9
- types of 16-7
- Documentation conventions
 - command-line Intro-9
 - icon Intro-8
 - typographical Intro-7
- Documentation notes Intro-13
- Documentation, types of
 - documentation notes Intro-13
 - error message files Intro-13
 - machine notes Intro-13
 - on-line manuals Intro-12
 - printed manuals Intro-12
 - release notes Intro-13
- DRAUTO parameter
 - description of 37-18
 - role in recovering from data-replication failure 29-18
 - set to RETAIN_TYPE 29-19
 - set to REVERSE_TYPE 29-19
- DRINTERVAL parameter
 - description of 37-19
 - setting for asynchronous updating 29-11
 - setting for synchronous updating 29-11
- DRLOSTFOUND parameter
 - description of 37-20
 - use with data replication 29-12
- DRTIMEOUT parameter
 - description of 37-20
 - role in detecting data replication failures 29-16
- dr.lostfound file 37-20
- DSA. *See* Dynamic scalable architecture.
- DSS applications. *See* Decision-support applications.
- DS_MAX_QUERIES parameter
 - changing with onmode 19-4
 - description of 37-21
 - estimating the value 19-9
 - use of 18-13
 - used by MGM 18-20
- DS_MAX_SCANS parameter
 - changing with onmode 19-4
 - description of 37-22

- use of 18-13
- used by MGM 18-20
- DS_TOTAL_MEMORY parameter
 - changing with onmode 19-4
 - deriving the value 37-24
 - description of 37-23
 - recalculated by server 37-24
 - use of 18-14
 - used by MGM 18-20
- DUMPCNT parameter 37-27
- DUMPCORE parameter 37-27
- DUMPDIR parameter
 - af.xxx assertion failure file A-4
 - and consistency checking 31-7
 - and shmenv file A-11
 - description of 37-28
 - gcore file A-6
- DUMPGCORE parameter 31-7, 37-28
- DUMPSHMEM parameter 31-7, 37-29
- Dynamic scalable architecture
 - advantages 10-3
 - description of 10-3

E

- End user
 - PDQ resource control 19-11
 - role in managing PDQ 19-9
- Environment configuration file A-7
- Environment variable
 - ARC_DEFAULT 3-32
 - ARC_KEYPAD A-11, A-13
 - DBNETTYPE 4-55
 - for GLS 3-14
 - for users of client applications 3-31
- INFORMIXDIR 3-12, 3-29, 4-54
- INFORMIXSERVER 3-12
- INFORMIXSHMBASE 12-11
- INFORMIXSQLHOSTS 4-25
- INFORMIXTERM 3-12
- ONCONFIG 3-12
- OPTCOMPIND 18-14, 19-10
- overview 3-11
- PATH 3-12, 3-29
- PDQPRIORITY 18-12

- required by application 3-12
- setting 3-13
- SQLEXEC 4-54
- SQLRM 4-55
- SQLRMDIR 4-55
- TBCONFIG A-9
- TERM 3-12
- TERMCAP 3-12
- Environment variables
 - INFORMIXCONCSMCFG 4-9
- en_us.8859-1 locale Intro-5
- Error message files Intro-13
- Error messages
 - for two-phase commit protocol 34-29
 - I/O errors on a chunk 31-10
 - /etc/hosts file 4-19
 - and client redirection 29-26
 - multiple residency 6-7
 - /etc/hosts.equiv file 4-21
 - /etc/passwd file, mentioned 4-21
 - /etc/services file 4-19
 - and client redirection 29-26
 - multiple residency 6-7
 - /etc/shadow file, mentioned 4-21
- Even distribution 16-9
- Event alarm
 - class id parameter 33-10
 - class message parameter 33-10
 - description 33-8
 - event severity codes 33-9
- Event alarms
 - mentioned 38-7
- Example
 - 7.2 relay module 4-53
 - connecting 7.2 client to 5.0 server 4-59
- DBSERVERALIASES and sqlhosts file 4-45
- DBSERVERALIASES
 - parameter 4-45
 - /etc/services file entry 4-20
 - how page cleaning begins 12-40
- IPX/SPX connection 4-50
- local loopback connection 4-48
- multiple connection types 4-50
- NETTYPE parameters for tuning 37-48

- relay module with three servers 4-55
- shared-memory connection 4-47
- SQLEXEC environment variable 4-54
- sqlhosts file 4-24
- TCP/IP connection 4-50
- Exclusive lock (buffer), description of 12-36
- exit codes, ontape utility 39-109
- Explicit temporary table 14-29
- Expression fragment
 - how searches are done 16-12
- Expression fragment, how searches are done 16-12
- Expression-based distribution scheme
 - and fragment elimination 16-12
 - arbitrary rule 16-11
 - description of 16-10
 - designing 16-31
 - hash rule 16-11
 - which type to use 16-31
 - with range rule 16-10
- Extensibility
 - description of 10-3
 - design principle 10-3
- Extensions
 - new data types 2-6
- Extent
 - automatic doubling of size 42-31
 - default size 42-27
 - description of 14-13
 - disk page types 42-28
 - how Universal Server allocates 14-14
 - information in sysextents table 38-18
 - initial size 14-13, 42-27
 - interleaving 14-32
 - key concepts concerning 14-14
 - merging 42-32
 - monitoring 33-64
 - next-extent allocation 42-31, 42-32
 - next-extent allocation strategies 42-33
 - next-extent size 14-13, 42-31
 - optimizing table 14-43

- procedure for allocating 42-31
- purpose of 14-13
- relationship to chunk 14-14
- size limitations 42-27
- size, for an attached index 16-20
- structure 14-13, 42-27
- tracking free pages using bit-map page 42-24
- validating consistency 31-5
- Extent size
 - index fragments 42-28
- External spaces
 - description of 1-8
 - in sysextspaces table 38-18
- External spaces. *See* extspace.
- Extspace
 - configuring 3-29
 - dropping using onspaces 15-26
- Extspaces
 - creating 15-25

F

- Fast recovery
 - description of 1-14, 26-3
 - details of process 26-5
 - effects of buffered logging 26-4
 - how Universal Server detects need for 26-4
 - mentioned 7-4, 9-8, 20-4
 - no logging 26-5
 - purpose of 26-4
 - role of checkpoint 12-58
 - role of PAGE_CHKPT reserved page 26-6
 - when needed 26-4
 - when occurs 26-3
- Fault tolerance
 - backups, and 1-14
 - data replication 1-15, 29-4, 29-6
 - fast recovery 1-14, 26-3
 - mirroring 1-15
- Feature icons Intro-8
- Features, product Intro-6
- File
 - af.xxx A-4
 - archive configuration file A-5
 - ARCrequid.NOT A-5
- buildsmi.xxx A-5
- configuration 3-11
- config.arc 6-8, 6-9, A-5
- connectivity configuration 4-18
- core A-5
- core.pid.cnt 37-29
- default configuration file A-9
- dr.lostfound 37-20
- /etc/hosts.equiv 4-21
- /etc/services 6-4
- /etc/shadow 4-21
- /etc/sqlhosts 6-4
- gcore 37-28
- gcore, xxx A-6
- in INFORMIXTMP directory A-7
- .informix A-6
- informix.rc environment file A-7
- .infos.dbservername A-8
- .inf.servicename A-7
- network security 4-21
- oncatlgr.out.pidnum A-8
- oncfg_servername.servnum 9-9, A-10
- onconfig A-9
- onconfig, during
 - initialization 9-5, 9-6
- onconfig.std
 - description of 3-11, A-9
 - during initialization 9-6
 - sample file A-14
- oper_deflt.arc 6-8, 6-9, A-11
- private environment file A-6
- .rhosts 4-21
- servicename.exp A-11
- servicename.str A-12
- shmем.pid.cnt 37-29
- shmем.xxx A-11
- sqlhosts 3-11, 4-24
- status_vset_volnum.itgr A-12
- summary of Universal Server files A-1
- sysfail.pidnum A-13
- tctermcap archive attributes file A-13
- template for configuration file A-9
- VP.servname.nnx A-13
- /etc/shadow 4-21
- File I/O. *See* Disk I/O.

FILLFACTOR parameter
 control how indexes fill 42-58
 description of 37-30

FLRU queues
 and reading a page from
 disk 12-43
 and releasing buffer 12-44
 description of 12-37
See also LRU queues.

Flushing
 buffers 12-45
 data-replication buffer, maximum
 interval 37-19
 of before-images 12-46

Forced residency
 initialization 9-10
 start/end with onmode 39-39

Forcing a checkpoint. *See*
 Checkpoint.

Foreground write
 and before-image 12-46
 description of 12-50
 monitoring 12-51, 33-25

formula
 memory grant 19-8

Forward pointer
 description of 42-39
 how it can become invalid 12-60
 role in a blobspace simple large
 object storage 42-68
 role in data storage 42-42
 role in dbspace simple large object
 storage 42-64
 unaffected by page
 compression 42-36

Fragment
 description of 16-3
 eliminating from search 16-12
 internal structure of 16-19
 monitoring disk usage 33-64,
 33-67
 monitoring I/O requests
 for 33-61
 nonoverlapping on multiple
 columns 16-15
 nonoverlapping on single
 column 16-13
 overlapping on single
 column 16-14

skipping 17-5
 skipping selected
 fragments 16-23
 skipping unavailable
 fragments 16-23
 turning DATASKIP ON or OFF
 for 39-74
 warning returned when skipped
 during query 37-13
See also Fragmentation.

FRAGMENT BY EXPRESSION
 clause, of CREATE
 TABLE 16-10

Fragment expression, arbitrary
 expression 16-11

Fragmentation
 and logging 16-7
 and mirroring 14-41
 and PDQ 16-25
 and rowids 16-20, 42-39
 choosing a distribution scheme
 for 16-29
 creating an explicit rowid
 column 16-21
 description of 16-3
 disk allocation for 16-19
 distribution schemes for 16-7
 equality search 16-12
 expressions, how evaluated 16-12
 for decision-support
 applications 16-25
 for finer granularity of archive
 and restore 16-28
 for improved availability 16-27
 for improved concurrency 14-41,
 16-26
 fragment elimination 16-12
 goals of 16-5, 16-23
 how many fragments 16-32
 internal structure of tables 16-19
 monitoring 17-6
 of temporary tables 16-7
 range search 16-12
 restrictions on index 16-18
 skipping fragments 17-4
 skipping inaccessible
 fragments 16-21
 smart large objects 16-9
 strategy 16-23

use of primary key 16-20
 with PDQ 16-5
See also Fragment.

Fragmentation strategy
 description of 16-3
 formulating 16-23

Fragmented table, using primary
 keys 42-40

Fragments
 index 42-28

Free list. *See* Chunk free list.

Free-map page
 description of blobspace free-map
 page 42-67
 role in blobspace logging 22-25

Functional index 42-60

G

Gateway transactions 37-30

Gateway, Informix, in
 heterogeneous commit 34-38

Gate, definition of 18-20

gcore
 file 31-7
 utility 37-27, 37-28

gcore.xxx file A-6

Global Language Support
 (GLS) Intro-5
 environment variables 3-14
 mentioned 3-27

Global pool, description of 12-33

Global transaction identification
 number 35-8

Group, parallel processing of 10-10

H

Hash table
 to buffer table 12-20
 to lock table 12-22

Hash tables 12-19

Heaps 12-31

Heterogeneous
 commit 34-38 to 34-46

Heterogeneous-commit
 transactions 37-30

HETERO_COMMIT configuration

parameter 37-30

Heuristic decision

result from independent
action 34-20

types of 34-21

Heuristic end-transaction

conditions for 34-26

description of 34-26

determining effects on

transaction 35-4

illustration, including logical log

records 34-35

messages returned by 34-28

results of 34-28

when necessary 34-26

Heuristic rollback

conditions resulting in 34-22

description of 34-22

illustration, including logical log

records 34-33

indications that rollback

occurred 35-5

results of 34-24

High availability data replication.

See Data replication.

Home page 42-37, **42-41**

Horizontal index link

described 42-56

how implemented 42-56

hostname

syntax rules 4-26

hostname field

multiple network interface

cards 10-38

using IP addresses 4-31

wildcard addressing 4-32

with IPX/SPX 4-35

with shared memory 4-30

with TCP/IP communication

protocol 4-30

Hot site backup. See Data

replication.

I

Icons

comment Intro-8

feature Intro-8

Identification parameters 3-17

Ill-behaved user-defined

routine 10-24

illsrta.xx file A-6

Inconsistency information

disk volume A-12

how to detect 31-3

Index

branch node 42-48

btree cleaner list 42-57

controlling how filled 42-58

delete flag 42-49

duplicate key values 42-54

effects on structure of item

insertion 42-57

functional 42-60

horizontal node links 42-56

how created and filled 42-50

how deleted items removed 42-57

insertion of indexed data 42-57

item described 42-49

key value locking 42-57

leaf node 42-48

location of index page 42-56

parallel building of 10-10

physical format and storage 42-55

repairing structures with oncheck

utility 39-5

reuse of freed pages 42-58

root node 42-48

structure of B+ tree 42-48

validating consistency 31-5

Index fragments

extent size 42-28

Index item

calculating the length of 42-59

defined 42-49

example of 42-50

how removed 42-57

merging 42-58

shuffling 42-58

when purged 42-49

Index page

compactness 37-30

contents 42-56

creation of first 42-51

effects of creation 42-51

effects of inserting item 42-57

page header 42-56

sibling pointers 42-56

size 42-56

structure 42-56

structure of 42-48

Industry standards, compliance

with Intro-14

Infinity item

and creation of branch node 42-53

defined 42-52

.informix file

mentioned 3-32, A-6

multiple residency 6-9

Informix recommendations

on allocation of disk space 14-9

on consistency checking 31-3

on mirroring the physical

log 24-8

INFORMIXCONCSMCFG

environment variable 4-9

\$INFORMIXDIR/etc/sqlhosts. See

sqlhosts file.

INFORMIXDIR environment

variable

and 5.x products 4-54

in shutdown script 3-30

in startup script 3-29

mentioned 3-12

multiple residency startup

script 6-9

INFORMIXDIR/bin

directory Intro-5

INFORMIXKEYTAB environment

variable

Environment variable

INFORMIXKEYTAB 4-9

INFORMIXSERVER environment

variable

during initialization 3-12

multiple residency startup

script 6-9

multiple versions of Universal

Server 3-30

- relation to
 - DBSERVERNAME 37-15
- use in client redirection 29-28
- with multiple residency 6-9
- INFORMIXSHMBASE
 - environment variable 12-11
- INFORMIXSQLHOSTS
 - environment variable 4-25
- INFORMIXSTACKSIZE
 - environment variable 12-31
- INFORMIXTERM environment variable 3-12
- INFORMIXTMP directory
 - description of A-7
 - servicename.exp file A-11
 - servicename.str file A-12
 - .inf.servicename file A-7
- INFORMIX-Universal Server
 - administering 3-6
 - administrator, description of 1-18
 - bad-sector mapping, absence of 1-19
 - client/server architecture 1-10
 - connecting to multiple servers 4-52, 5-3
 - description of 1-3
 - distributed queries 1-16
 - external spaces, description of 1-8
 - fault-tolerant features 1-13
 - features beyond the scope of 1-18
 - fragmentation 1-13
 - high performance of 1-11
 - management of data 14-7
 - message-log file 37-46
 - multimedia support 1-7
 - multiple instances 5-3
 - onmode utility 39-46
 - parallel database query
 - feature 1-13
 - profile statistics 33-20
 - raw-disk management 1-12
 - resident portion of shared memory 37-60
 - scalability of 1-11
 - security 1-16
 - shut down with UNIX script 3-30
 - Simple large object compression, absence of 1-19
 - testing environment 5-3

- types of applications 18-6
- types of users 1-17
- user-defined routines, description of 1-8
- users 1-17
- informix.rc environment file
 - description of A-7
 - multiple residency 6-9
 - use of 3-32
- .infos.dbservername file
 - description of A-8
 - regenerate 39-45
- .inf.servicename file A-7
- Inheritance
 - type and table, description of 1-8
 - type and table, mentioned 1-4
- Initial configuration
 - creating blobspaces and dbspaces 3-28
 - disk layout 14-36
 - guidelines for root dbspace 14-21
 - raw disk devices versus cooked files 14-6
- Initialization
 - checkpoint 9-9
 - commands 15-8
 - configuration changes 9-9
 - configuration files 9-5
 - control returned to user 9-10
 - conversion of internal files 9-8
 - disk space 3-28, 9-3, 9-8
 - disk space for multiple residency 6-7
 - disk structures initialized 42-4
 - environment variables to set 3-11
 - fast recovery 9-8
 - forced residency 9-10
 - message log 9-10
 - oncfg_servername.servernum file 9-9
 - onconfig file 9-6
 - onconfig.std file 9-6
 - reserved page information 42-8
 - shared memory 9-3
 - SMI tables 9-10
 - steps in 9-4
 - temporary tablespaces 9-9
 - upon completion 9-11

- utilities for 9-4
- virtual processors 9-8
- Installation
 - definition of 3-7
 - getting ready 3-8
 - private, creating 3-9
 - upgrading from INFORMIX-OnLine 3-8
 - when no other Informix products are present 3-7
 - when other Informix products are present 3-8
 - when SE is present 3-8
- Installing
 - private server 3-9
- Integrity, data. *See* Consistency checking.
- Interface attributes for ON-Archive A-13
- Interprocess communication
 - in nettype field 4-28
 - shared memory for 12-5
- Interval, checkpoint 37-11
- IP address
 - how to find 4-31
 - use in hostname field 4-31
- ipcsbm protocol and communications portion (shared memory) 12-33
- IPC. *See* Interprocess communications.
- IPX/SPX
 - in hostname field 4-35, 4-50
 - in nettype field 4-28
 - in servicename field 4-38
 - multiple residency 6-7
 - service, definition of 4-38
 - sqlhosts entry 4-50
- ISAM calls tabulated 39-97
- ISO 8859-1 code set Intro-5
- Isolation level
 - Committed Read and simple large objects 12-60
 - Dirty Read and simple large objects 12-60
- I/O errors during processing 31-10
- I/O. *See* Disk I/O.

J

Join method, influence of
 OPTCOMPIND 19-10
Join, parallel processing of 10-10

K

KAIO thread 10-29
Keep-alive option, in options
 field 4-40
Kernel asynchronous I/O
 description of 10-29
 non-logging disk I/O 10-26
Key value
 and index items 42-49
 checking ordering with
 oncheck 39-11, 39-19
 duplicates 42-54, 42-55
 for shared memory 12-13
 locking 42-57
keytab file
 purpose of 4-9

L

Large objects
 storage, mentioned 1-4
Latch
 and wait queue 10-18
 description of 12-34
 displaying information with
 onstat 39-80, 39-100
 identifying the resource
 controlled 39-100
 monitoring statistics and
 use 33-27
 mutex 10-18, 12-34
Level-0 dbspace
 use in consistency checking 31-8
Lightweight processes 10-6
Limits
 on chunk size 14-6
Linking, name of root
 dbspace 37-61
LIO virtual processors
 description of 10-27
 how many 10-27

Listen threads
 and multiple interface
 cards 10-38
 description of 10-33
Load balancing
 as performance goal 14-36
 done by virtual processors 10-9
 of critical media 14-38
 through use of
 DBSPACETEMP 14-38
Local loopback
 compared with shared-memory
 connection 4-15
 connection 4-14, 10-31
 example 4-48
 restriction 4-14
Locale Intro-5
Lock
 and oncheck 39-7
 and wait queue 10-18
 buffer-access-level flag bits 39-84
 description of 12-34
 information in syslocks
 table 38-19
 key value locking 42-57
 maximum time to acquire 37-18
 monitoring 33-29, 39-79, 39-92
 type codes 39-92
 types 12-35
Lock table
 contents of 12-22
 hash table 12-22
 maximum number of
 entries 12-22
Lock-access level, of a buffer 12-43
Locking
 for multiprocessor 37-46
 when occurs 12-43
 when released 12-44
LOCKS parameter
 description of 37-32
 mentioned 3-21
LOGBUFF parameter
 and logical log buffers 12-27
 description of 37-33
 mentioned 3-18
 smart large objects, and 12-64

LOGFILES parameter
 changing 23-10
 description of 37-34
 mentioned 3-18
 use in logical-log size
 determination 22-8
Logging
 activity that is always logged 20-6
 definition of transaction
 logging 20-9
 effect of buffering on logical log
 fill rate 22-20
 monitoring activity 33-48
 physical logging
 description of 24-3
 process of 24-9
 purpose of 24-4
 sizing guidelines for 24-5
 suppression in temporary
 dbspaces 14-22
 process for blobspace data 22-28
 process for dbspace data 22-26
 role in data replication 29-9
 role of blobspace free-map
 page 22-29, 42-67
 suppression for implicit
 tables 14-22
 Universal Server processes
 requiring 20-4
 when to buffer transaction
 logging 20-12
 when to use transaction
 logging 20-12
 with simple-large-object
 data 20-7
 with smart-large-object data 20-8
 See also Database-logging status.
Logical consistency, description
 of 26-5
Logical log
 administration tasks for
 blobspaces 22-23
 backing up 22-8
 checking consistency 39-20
 configuration parameters 3-18,
 23-10
 description of 12-26, 20-4, 22-3
 determining disk space
 allocated 22-8

- how to prevent logs full 22-8
- in root dbspace 42-4
- maximum number of files 37-36
- monitoring for fullness using
 - onstat 33-48
- monitoring with SMI 38-20
- optimal storage of 14-38, 14-39
- preserving space in 22-8
- purpose of 1-14
- setting high-water marks 22-21
- size, guidelines 22-7
- size, performance
 - considerations 22-5
- types of records 12-26
- See also* Logical-log buffer, Logical-log file.
- Logical page number 42-38
- Logical recovery
 - number of threads 37-52
 - role in data replication 29-9
- Logical units of storage
 - description of 14-16
 - list of 14-4
- Logical volume manager (LVM)
 - description of 14-49
 - mirroring alternative 27-6
- Logical-log buffer
 - and data-replication buffer 12-28
 - and LOGBUFF parameter 37-33
 - and logical-log records 12-52
 - description 12-26
 - description of 12-26
 - flushing 12-52
 - flushing for non-logging
 - databases 12-54
 - flushing when a checkpoint occurs 12-54
 - flushing when no before-image 12-54
 - flushing with unbuffered logging 12-53
 - monitoring 33-53
 - role in logging process 20-10
 - synchronizing flushing 12-45
 - when flushed to disk 12-52, 22-28
 - when it becomes full 12-52

- Logical-log file
 - adding a log file
 - using ON-Monitor 23-4
 - using onparams 23-5
 - allocating disk space for 22-5
 - and reserved pages 42-9
 - backup
 - effect on performance 22-5
 - goals of 22-16
 - to free deleted blobpages 22-24
 - changing the size of 23-9
 - consequences of not freeing 22-17
 - created during
 - initialization 37-34
 - description of 20-4, 22-4
 - displaying contents 39-30
 - dropping a log file
 - using ON-Monitor 23-7
 - using onparams 23-7
 - file status 22-14
 - how to free 23-13
 - how to switch 23-15
 - I/O to 10-26
 - location 22-13
 - logid number 22-13
 - mirroring a dbspace containing a file 27-7
 - moving to another dbspace 23-8
 - number of files 22-12
 - rate at which files fill 22-19
 - reading the log file 39-30
 - relationship between unique ID and logid 22-13
 - reserved page information 42-9
 - role in fast recovery 26-5, 26-7 to 26-8
 - size 22-12, 37-35
 - status flags 22-14
 - switch using onmode 39-40
 - switching to activate blobspaces
 - chunks 22-23
 - switching to activate blobspaces 22-23
 - unique ID number 22-13
 - when freed for reuse 22-17
 - when Universal Server tries to free files 22-17
- See also* Logical log.

- Logical-log I/O virtual processors 10-27
- Logical-log record
 - additional columns of 41-7
 - displaying 39-30
 - flushing under two-phase commit protocol 34-32
 - for a checkpoint 41-5
 - for a drop table operation 41-4
 - generated by a rollback 41-4
 - header columns 41-6
 - involved in distributed transactions 41-5
 - involved in two-phase commit protocol 34-30, 41-5
 - role in fast recovery 26-7, 26-7 to 26-8
 - role in two-phase commit protocol 34-6
 - SQL statements generating 20-7
 - types 41-7
- Universal Server processes
 - requiring 20-4
 - when written to logical-log buffer 22-27
- logid 22-13
- Logs-Full High-Water Mark 22-8
- LOGSIZE parameter
 - changing 23-10
 - description of 37-35
 - mentioned 3-18
 - use in logical log size determination 22-8
- LOGSMAX parameter
 - changing 23-12
 - description of 37-36
 - mentioned 3-18, 22-12
- Long transaction
 - consequences of 22-18
 - description of 22-19
 - mentioned within two-phase commit discussion 34-19, 34-24, 34-27
 - preventing development of 22-19
 - starting percentage 37-41, 37-42

- LRU queues
 - and buffer pool
 - management 12-37
 - and buffer table 12-25
 - components 12-36
 - composition of 12-36
 - description of 12-36
 - displaying information with
 - onstat 39-79, 39-99
 - FLRU queues 12-37
 - how Universal Server
 - selects 12-38
 - MLRU queues 12-37
 - modified pages, percentage
 - of 37-37, 37-38
 - ordering of 12-37
 - rationale for ordering 12-37
 - reason for multiple 12-38
 - recommendation on
 - configuring 12-38
- LRU write
 - description of 12-51
 - monitoring 33-25
 - triggering of 12-51
 - who performs 12-51
- LRUS configuration parameter
 - setting the value of 12-38
- LRUS parameter
 - description of 12-36, 37-37
 - mentioned 3-22
- LRU_MAX_DIRTY parameter
 - and LRU_MIN_DIRTY
 - parameter 12-36
 - description of 37-37
 - example of use 12-40
 - how to calculate value 12-40
 - mentioned 3-22
 - purpose of 12-39
 - role in buffer-pool
 - management 12-40
- LRU_MIN_DIRTY parameter
 - and LRU_MAX_DIRTY
 - parameter 12-36
 - default value 12-40
 - description of 37-38
 - example of use 12-40
 - how to calculate value 12-40
 - mentioned 3-21

- role in buffer pool
 - management 12-40
 - when tested 12-49
- LTAPEBLK parameter
 - description of 37-38
 - mentioned 3-20
- LTAPEDEV parameter
 - mentioned 3-20
 - setting 3-15
- LTAPESIZE parameter
 - mentioned 3-20
- LTXEHWM parameter
 - and physical log 24-8
 - changing 23-12
 - description of 37-41
 - mentioned 3-18
 - role in heuristic rollback 34-22
 - role in preventing long
 - transactions 22-22
- LTXHWM parameter
 - changing 23-12
 - description of 37-42
 - flushing of logical-log
 - buffer 12-53
 - mentioned 3-18
 - role in heuristic rollback 34-22
 - role in preventing long
 - transactions 22-21

M

- Machine notes Intro-13
- Main_loop thread
 - role in checkpoint 12-58
 - role in two-phase commit
 - recovery 34-12
- Main_loop() thread 12-57
- Major features Intro-6
- Managing physical disk I/O 14-3
- Manual recovery
 - deciding if action needed 35-10
 - determining if data
 - inconsistent 35-6
 - example of 35-11
 - obtaining information from
 - logical log files 35-7
- procedure to determine if
 - necessary 35-3
 - use of GTRID 35-8
- Mapping, bad sector 1-19
- MAX_PDQPRIORITY parameter
 - changing with onmode 19-4
 - description of 37-43
 - limiting PDQ priority 19-4
 - maximizing PDQ 19-6
 - setting with OLTP 19-5
 - use of 18-13
- Media failure
 - detecting 27-10
 - recovering from 27-5
 - restoring data 1-14
- Memory
 - specifying size of 37-54
- Memory Grant Manager
 - description of 19-12
 - using with the onstat utility 19-12
- Memory. *See* Shared memory.
- mentioned 12-7
- Message log
 - alphabetical listing of
 - messages 40-3
 - and data corruption 31-9
 - and event alarms 33-8
 - categories of messages 40-4
 - description of 33-6, 40-3
 - displaying with onstat
 - utility 39-79, 39-94
 - during initialization 9-10
 - file pathname 37-46
 - location of 37-46
 - mentioned A-8
 - monitoring 33-7, 33-8
- Messages, for onspaces 40-31
- MGM. *See* Memory Grant Manager.
- Migration. *See* Informix Migration Guide.
- Mirror chunk
 - adding 28-9
 - changing status of 28-10
 - creating 28-6
 - disk reads from 27-9
 - disk writes to 27-9
 - pathnames 42-13
 - recovering 27-10, 28-10

- relinking after disk failure 28-12
- structure 27-11, 42-16
- MIRROR** parameter
 - changing 28-4
 - description of 28-4, 37-44
 - initial configuration value 28-4
 - mentioned 3-17
- Mirroring**
 - activity during processing 27-9
 - alternatives 27-6
 - and chunk table 12-20
 - and multiple residency 6-6
 - asynchronous write requests 27-9
 - benefits of 27-4
 - changing chunk status 28-10
 - costs of 27-5
 - creating mirror chunks 28-6
 - description of 1-15, 27-4
 - detecting media failures 27-10
 - during system initialization 28-7
 - effects of 3-18
 - enable flag 37-44
 - enabling 28-4
 - ending 28-12
 - if the dbspace holds logical-log files 27-7
 - initial chunk 37-45
 - network restriction 27-4
 - recommended disk layout 14-41
 - recovering a chunk 28-10
 - recovery activity 27-8
 - reserved page information 42-13
 - split reads 27-9
 - starting 28-3, 28-6
 - status flags 27-8
 - steps required 28-3
 - what happens during
 - processing 27-9
 - when mirroring begins 27-7
 - when mirroring ends 27-11
- MIRROROFFSET** parameter 14-21
 - and multiple residency 5-4
 - description of 37-45
 - mentioned 3-17
 - setting 28-7
 - when needed 15-7
- MIRRORPATH** parameter
 - and multiple residency 5-4
 - description of 37-45
- mentioned 3-17, 14-21
- multiple residency 6-6
- setting 28-7
- specifying a link pathname 37-45
- Miscellaneous (MSC) virtual processor** 10-39
- Mixed transaction result** 34-25
- MLRU queues**
 - and flushing of regular buffers 12-46
 - and LRU_MIN_DIRTY parameter 12-40
 - description of 12-37
 - how buffer is placed 12-38
 - how to end page-cleaning 12-40
 - limiting number of pages 12-39
 - role in buffer modification 12-45
 - when cleaning ends 12-40
 - See also* LRU queues.
- Mode**
 - current operating mode 8-4
 - description of 7-3
 - graceful shutdown 8-5
 - immediate shutdown 8-6
 - list of mode changes 8-3
 - off-line from any mode 8-7
 - off-line to on-line 8-4
 - off-line to quiescent 8-3
 - on-line to quiescent, gracefully 8-5
 - on-line to quiescent, immediately 8-6
 - quiescent 7-4
 - quiescent to on-line 8-5
 - reinitializing shared memory 8-3
 - taking off-line 8-7
 - users permitted to change 8-3
- MODE ANSI keywords, and database logging status** 20-11
- Monitoring**
 - bspaces 33-67
 - smart large objects 33-67
- Monitoring Universal Server**
 - active tablespaces 33-31
 - blob space storage efficiency 15-22
 - buffer-pool activity 33-24
 - buffers 12-25, 33-21
 - checkpoints 33-17
 - chunk status 33-55
 - chunks 33-60
 - configuration parameter values 33-15
 - databases 33-47
 - data-replication status 33-76
 - disk I/O queues 10-30
 - extents 33-64
 - fragment load 33-61
 - fragmentation disk use 33-64, 33-67
 - latches 33-19, 33-27
 - list of tools 3-7
 - list of topics 33-5
 - locks 33-29
 - logging status 33-47
 - logical-log buffers 33-53
 - logical-log files 33-48
 - message log 33-7
 - PDQ resources 33-40
 - physical-log buffer 12-27, 33-53
 - physical-log file 33-51
 - profile of activity 33-20
 - sessions 33-35
 - shared memory 33-19
 - shared-memory segments 33-19
 - Simple large objects in
 - bspaces 33-74
 - sources of information 33-6
 - stack size 33-38
 - threads 33-35
 - transactions 33-45
 - user threads 12-24
 - using oncheck 33-13
 - using ON-Monitor 33-12
 - using onperf 33-13
 - using onstat 33-13
 - using SMI tables 33-13
 - virtual processors 33-32
- Monitoring, length of disk I/O queues** 10-30
- MSGPATH** parameter
 - and multiple residency 5-4
 - description of 37-46
 - mentioned 3-20
 - multiple residency 6-6
 - setting 3-15
- Multiple concurrent threads (MCT)** 10-12

- Multiple connection types
 - example 4-50
 - in sqlhosts file 4-45
 - See also* Connection.
- Multiple instances of Universal
 - Server on same computer 5-3
- Multiple network interface
 - cards 10-38
- Multiple residency
 - and blobspace 6-7
 - and chunk assignment 6-7
 - and dbspaces 6-7
 - and multiple binaries,
 - warning 6-4
 - archiving 6-8
 - backups 6-8
 - benefits of 5-3
 - configuration file 5-3
 - configuration parameters 5-4
 - DBSERVERNAME parameter 6-5
 - definition of 3-10, 5-3
 - editing the ONCONFIG file 6-5
 - /etc/hosts & /etc/services 6-7
 - example 4-52
 - 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 6-6
- MSGPATH parameter 6-6
- ONCONFIG environment
 - variable 5-4
- planning for 6-3
- requirements 6-3
- ROOTNAME parameter 6-6
- ROOTPATH parameter 6-6
- SERVERNUM parameter 6-4, 6-5
- sqlhosts file 6-7
- startup script 6-9
- steps for preparing 6-4
- to isolate applications 5-3
- use for testing 5-4

- Multiprocessor computer
- advantages on 10-4
- AFF_SPROC parameter 37-8

- MULTIPROCESSOR
 - parameter 10-20
- processor affinity 10-11, 37-79
- MULTIPROCESSOR parameter
 - description of 10-20, 37-46
 - for single-processor
 - computer 10-20
- Multi-threaded database server. *See*
 - Dynamic scalable architecture.
- Multithreaded processes,
 - description of 10-6
- Multithreading, use of OS
 - resources 10-9
- Mutex
 - description of 10-18, 12-34
 - on buffer table hash table 12-43
 - synchronization 10-18
 - when used 12-34

N

- NCPUVPS parameter
 - and VPCLASS 37-51
- nettype field
 - description of 4-27
 - format of 4-27
 - summary of values 4-29
 - syntax of 4-27
 - use of interface type 4-50
- NETTYPE parameter
 - and multiple network
 - addresses 10-38
 - description of 37-47
 - mentioned 4-44
 - ON-Monitor screen entries 11-5
 - poll threads 10-32
 - purpose of 4-44
 - role in specifying a protocol 10-32
 - tuning example 37-48
 - vp class entry 10-32
- NetWare file server 4-35
- Network communication
 - connection types 10-31
 - using IPX/SPX 4-35, 4-38, 4-50
 - using TCP/IP 4-30, 4-36
- Network connection
 - how implemented 10-33
 - types of 10-31
 - when to use 4-13
- Network Information Service 4-19
- Network interface cards
 - and listen threads 10-38
 - sqlhosts file 10-38
 - using multiple 10-38
- Network protocols,
 - specifying 10-32
- Network security
 - /etc/passwd 4-21
 - /etc/shadow 4-21
 - /etc/hosts.equiv 4-21
 - files 4-21
 - .rhosts file 4-21
- Network virtual processors
 - and poll threads 10-32
 - description of 10-31
 - how many 10-33
- NIS servers, effect on /etc/hosts
 - and /etc/services 4-20
- NOAGE parameter
 - and VPCLASS 37-50
 - description of 37-49
 - purpose of 10-21
- Node, index (disk)
 - branch node defined 42-50
 - checking horizontal and vertical
 - nodes 39-11, 39-19
 - contents of leaf nodes 42-52
 - creation of branch nodes 42-53
 - defined 42-50
 - leaf node defined 42-50
 - pointer 42-52
 - root node 42-51
 - root node defined 42-50
 - types of 42-50
 - what branch nodes point to 42-54
 - when root node fills 42-51
- Non-Informix databases 1-16
- Non-yielding virtual
 - processor 10-25
- Notification file A-5
- NUMAIOVPS parameter
 - and VPCLASS 37-50
 - description of 37-50
 - purpose of 10-29

- Number of
 - page-cleaner threads 37-12
- NUMCPUVPS parameter
 - and poll threads 10-33
 - description of 37-51
 - purpose of 10-19
 - used by
 - DS_TOTAL_MEMORY 37-24

O

- Object-relational database-
management system 1-3
- Off-line mode 7-3
- Offset
 - definition of 14-9
 - purpose of 15-6
 - use in prevention of overwriting
 - partition information 14-9
 - use in subdividing partitions 14-9
 - when needed 15-6
- OFF_RECVR_THREADS
 - parameter, description of 37-52
- OLTP application
 - and fragmentation 16-26
 - characteristics of 18-6
 - definition of 18-6
 - uses of 18-6
- ON-Archive
 - backing up logical-log files 22-16, 22-18
 - configuration file 3-27
 - config.arc file 3-27, A-5
 - estimating disk space for 14-35
 - mentioned 3-19
 - modifying database logging
 - status 21-5
 - use in starting data
 - replication 30-10
- ON-Archive, fatal error
 - record A-13
- oncatlgr utility
 - in UNIX startup script 3-30
 - message file A-8
 - multiple residency 6-8
- oncatlgr.out.pidnum file A-8
- oncfg_servername.servnum
file 9-9, A-10

- oncheck option descriptions 39-11
- oncheck utility
 - and locking 39-7
 - check-and-repair options 39-5
 - comparison with onstat 33-13
 - corrective actions 31-4
 - description of 39-5
 - list of functions 39-6
 - monitoring sbspaces 33-67
 - obtaining blob space
 - information 33-73
 - obtaining chunk
 - information 33-56, 33-62
 - obtaining configuration
 - information 33-16
 - obtaining extent
 - information 33-64
 - obtaining fragmentation
 - information 33-65
 - obtaining logical-log
 - information 33-50
 - obtaining physical-log
 - information 33-52
 - obtaining simple large object-
storage statistics 33-72
 - obtaining tblspace
 - information 33-66
 - options
 - cc 39-17
 - cd 39-17
 - ce 39-11, 39-19
 - ci 39-20
 - ci 39-19
 - cr 39-20
 - pB 15-22, 39-21
 - pD 39-22
 - pd 39-21
 - pe 42-5
 - pK 39-22
 - pk 39-22
 - pL 39-22
 - pl 39-22
 - pP 39-23
 - pp 39-23
 - pr 39-23
 - pT 39-24
 - pt 39-24
- overview of functionality 39-5

- Simple-large-object storage
 - information 15-22
 - suppressing messages 39-15
 - syntax 39-8
 - use in consistency checking 31-4
- ONCONFIG configuration file
 - conventions 37-6
 - conventions used 37-5
 - description 37-6, A-9
 - during initialization 9-5, 9-6
 - editing with multiple
 - residency 6-5
 - mentioned 4-44
 - multiple residency 4-52, 6-4
 - parameters 4-44
 - relation to
 - INFORMIXSERVER 3-12
 - white space 37-6
- ONCONFIG environment variable
 - and ONCONFIG file A-10
 - changes for multiple
 - residency 6-5
 - interaction with TBCONFIG A-10
 - mentioned 3-12
 - multiple residency startup
 - script 6-9
 - multiple versions of Universal
Server 3-30
 - use with multiple residency 5-4
- onconfig file A-9
- ONCONFIG file parameters. *See*
Configuration parameter.
- ONCONFIG parameter, and
multiple residency 6-4
- onconfig.std file
 - and multiple residency 6-4
 - description A-9
 - during initialization 9-6
 - sample A-14
 - when installed 3-11
- ondblog utility 39-25
- oninit utility
 - bringing Universal Server on-
line 8-4, 15-9
 - initializing Universal Server 8-4
 - option descriptions 39-29
 - p option 9-9, 14-29
 - starting Universal Server 39-27
 - temporary tables 14-29, 39-28

- On-line manuals Intro-12
- On-line mode, description of 7-4
- On-line transaction processing. *See* OLTP application.
- On-line transaction processing. *See* PDQ.
- onload utility. *See Informix Migration Guide.*
- onlog utility
 - description of 39-30
 - filters for displaying logical-log records 39-33
 - filters for reading logical-log records 39-32
 - reconstructing a global transaction 35-7
- onmode utility
 - adding a shared-memory segment 39-43
 - bringing Universal Server on-line 8-5
 - changing PDQ parameters 19-4
 - changing shared-memory residency 13-14, 39-39
 - changing Universal Server mode 39-37
 - description of 39-35
 - dropping CPU virtual processors 11-10
 - forcing a checkpoint 13-16, 39-39
 - freeing a logical-log file 23-14, 23-15
 - graceful shutdown 8-6
 - immediate shutdown 8-7
 - killing a participant thread 34-19
 - killing a session 34-22, 39-40
 - killing a transaction 34-19, 34-28, 35-4, 39-41
 - options
 - c force checkpoint 39-39
 - d set data replication type 39-42
 - I B-1
 - k take off-line 39-37, 39-38
 - l switch logical-log file 39-40
 - n end forced residency 39-39
 - p add or remove virtual processor 39-45
 - r begin forced residency 39-39
 - R regenerate .infos file 39-45
 - s take to quiescent 39-37, 39-38
 - Z kill transaction 39-41
 - set decision-support parameters 39-46
 - setting database server type 30-10, 30-20, 39-42
 - switching logical-log files 23-15, 39-40
 - take Universal Server off-line 8-8
 - trapping errors B-1
 - user thread services onmode utility requests 10-6
- ON-Monitor
 - access and use 36-3
 - adding a logical-log file 23-4
 - adding mirror chunks 28-9
 - archive menu and options 36-10
 - data-replication screen 36-15
 - dbspaces menu and options 36-9
 - diagnostics screen 36-16
 - dropping a logical-log file 23-7
 - enabling mirroring 28-5
 - force-ckpt option 36-9, 36-10
 - help 36-4
 - initialization screen 36-12
 - logical-logs menu and options 36-11
 - mode menu and options 36-10
 - parameters menu and options 36-7
 - parameters screens 36-12 to 36-17
 - PDQ screen 36-17
 - performance screen 36-14
 - recovering a chunk 28-11
 - setting performance options 13-12
 - setting shared memory parameters 13-7, 13-10
 - setting virtual processor parameters 11-4
 - starting mirroring 28-8
 - status menu and options 36-7
 - taking a chunk down 28-10
- onparams utility
 - adding a logical-log file 23-5, 39-51
 - changing physical log size, location 39-53
 - changing physical-log location 25-5
 - changing physical-log size 25-5
 - description of 39-50
 - dropping a logical-log file 23-7, 39-52
- onperf utility 33-13
- Onspaces utility
 - Df and CREATE TABLE 39-62
- onspaces utility
 - adding a chunk 39-64
 - adding a chunk to a sbspace 39-65
 - adding a mirror chunk 28-9
 - changing chunk status 39-72
 - creating a blobspace, dbspace, extspace, or temporary dbspace 39-56
 - creating an sbspace 39-59
 - creating an sbspace using -Df 39-61
 - dropping a blobspace, dbspace, extspace or sbspace 39-63
 - dropping a chunk 39-67
 - ending mirroring 28-13, 39-71
 - example modifying DATASKIP 17-5
 - f option 17-5
 - modifying DATASKIP parameter 17-5
 - recovering a down chunk 28-11
 - specifying unavailable fragments 16-28
 - starting mirroring 28-9, 39-69
 - taking a chunk down 28-11
- onstat utility
 - and CPU virtual processors 10-20
 - comparison with oncheck utility 33-13
 - description of 39-75
 - freeing blobpages and timing 39-86
 - g mgm option 19-12
 - header 39-81
 - monitoring blobspace 33-69
 - monitoring buffer use 12-25, 33-21, 33-22, 33-23
 - monitoring buffer-pool 33-25, 33-26
 - monitoring byte locks 12-22

- monitoring checkpoints 33-17
- monitoring chunk status 33-55
- monitoring configuration 33-15
- monitoring data replication 33-76
- monitoring disk usage 33-60, 33-61
- monitoring fragment load 33-61
- monitoring latches 33-27, 33-28
- monitoring locks 33-29, 33-30
- monitoring log buffers 33-53
- monitoring logical-log files 33-49
- monitoring PDQ 33-40
- monitoring physical log 33-51
- monitoring sessions 33-35
- monitoring shared memory 33-19
- monitoring tbspaces 33-33
- monitoring the message log 33-8
- monitoring transactions 33-45
- monitoring Universal Server profile 33-20
- monitoring virtual processors 33-32, 33-34
- option descriptions 39-17, 39-82
- options
 - a 39-83
 - b 39-83
 - c 39-78
 - D 39-87
 - d 39-84, 39-86
 - F 39-87
 - k 39-92
 - l 39-93
 - m 39-79, 39-94
 - o 39-79
 - p 39-96
 - R 39-99
 - r 39-79
 - s 39-100
 - t 39-101
 - u 39-102
 - X 39-106
 - z 39-106
 - 39-78, 39-82
 - (none) 39-82
- repeated execution with -r 39-79
- repeated execution with seconds parameter 39-81
- syntax 39-77

- table of options and functions 39-75
- terminating interactive mode 39-91
- terminating repeating sequence 39-91
- tracking a global transaction 34-29
- using SMI tables for onstat information 38-33
- using with shared-memory source file 39-80
- ontape utility
 - backing up logical-log files 22-16, 22-18
 - data replication functions 39-109
 - description of 39-107
 - exit codes 39-109
 - LTAPEBLK, use of 37-38
 - mentioned 3-19
 - modifying database logging status 21-7
 - tasks performed by 39-107
 - use in starting data replication 30-10
- onunload utility
 - LTAPEBLK, use of 37-38
- onunload utility. *See Informix Migration Guide.*
- ON_RECVRY_THREADS parameter
 - description of 37-52
 - role in data replication 29-14
- Opaque data types
 - mentioned 1-6
- OPCACHEMAX parameter
 - description of 37-54
- Operating system, scheduled jobs 3-31
- Operating Universal Server, things to avoid 32-3
- Operating-system files. *See Cooked file space.*
- Operating-system job to change PDQ parameters 19-5
- oper_deflt.arc file
 - and multiple residency 6-8
 - contents A-11

- OPTCOMPIND environment variable 19-10
 - use of 18-14
 - See also* OPTCOMPIND parameter.
- OPTCOMPIND parameter
 - choosing a value 19-10
 - description of 37-54
 - influencing the optimizer 19-10
 - use of 18-14
 - See also* OPTCOMPIND environment variable.
- OPTCOMPIND value, managing 19-10
- Optical cache
 - number of kilobytes 33-71, 39-95
 - number of kilobytes of simple large objects written 33-71, 39-96
 - number of kilobytes of simple large objects written to staging area 33-71, 39-96
 - number of simple large objects 33-71, 39-95
 - number of simple large objects written 33-71, 39-95, 39-96
 - number of simple large objects written to staging area 33-71, 39-96
 - session id for user 33-71, 39-96
 - userid of client 33-71, 39-96
- Optical memory cache
 - allocation 33-70, 39-95
 - availability of memory 33-70, 39-95
 - described 33-70, 39-79, 39-95
 - number of simple large objects 33-70, 39-95
 - size 33-70, 39-95
- Optical storage and STAGEBLOB parameter 37-70
- Optical (OPT) virtual processor 10-39
- Optimizing, hash joins vs. nested-loop joins 37-55
- options field
 - buffer-size option 4-42
 - keep-alive option 4-40

- list of options 4-38
- security option 4-40
- syntax rules 4-39
- ORDBMS. *See* Object-relational database management system.
- Outer join, effect on PDQ 18-19

P

Page

- bit-map page 42-67
- blobpage 42-67
- blobpage free-map page 42-67
- components of dbpage
 - page 42-34
- compression 42-38, 42-46
- dbpage page types 42-28
- dbpage simple-large-object
 - page 42-65
- definition of full page 42-41
- description of 14-10
- determining Universal Server
 - page size 15-15
- displaying contents with
 - oncheck 39-23
- free page, definition of 42-29
- fullness bit values 42-25
- fullness, 4-bit values 42-26
- header components 42-35
- least-recently used 12-37
- locating in shared memory 12-43
- logical page number 42-38
- most-recently used 12-37
- page types in extent 42-28
- physical page number 42-38
- relationship to chunk 14-10
- size recorded in reserved
 - pages 42-8
- slot table 42-35
- structure and storage of 42-34
- validating consistency 31-6
- Page compression 42-36, 42-38, 42-46
- Page size
 - shown in ON-Monitor 13-8
 - shown with onstat -b 39-84
 - stored in reserved pages 42-8

- Page-cleaner table
 - description of 12-22
 - number of entries 12-22
- Page-cleaner threads
 - alerted during foreground
 - write 12-51
 - codes for activity state 39-88
 - description of 12-45
 - flushing buffer pool 12-45
 - flushing of regular buffers 12-46
 - monitoring 12-23
 - monitoring activity 39-78, 39-87
 - number of 37-12
 - role in chunk write 12-52
 - sleeping forever 12-50
- PAGE_1CKPT reserved page 42-7
- PAGE_1PCHUNK reserved
 - page 42-11
- PAGE_2CKPT reserved page 42-7
- PAGE_ARCH reserved page 42-8, 42-14
- PAGE_CKPT reserved page 42-8
- PAGE_CONFIG reserved
 - page 39-20, 42-8
 - mentioned 9-6, 9-9
- PAGE_DBSP reserved page 42-11
- PAGE_MCHUNK reserved
 - page 42-13
- PAGE_PCHUNK reserved
 - page 42-11
- PAGE_PZERO reserved page
 - contents of 42-8
 - mentioned 9-8
 - when written to 42-7
- Parallel Database Query. *See* PDQ.
- Parallel processing
 - mentioned 1-13
 - virtual processors 10-10
- Parallel sort
 - interaction with PDQ 18-21
 - memory allocation 12-32
- Parameters
 - setting decision-support with
 - onmode 39-46
 - specifying dataskip with
 - onspaces 39-74
 - used with PDQ 18-11

- Participant database server
 - automatic recovery 34-14
 - description of 34-6
- Partnum field in systables 42-20
- PATH environment variable
 - in shutdown script 3-30
 - in startup script 3-29
 - mentioned 3-12
 - multiple residency startup
 - script 6-9
- PDQ (Parallel Database Query)
 - administrator control of
 - resources 19-11
 - and correlated subqueries 18-19
 - and remote tables 18-19
 - and stored procedures 18-18
 - and triggers 18-18
 - controlling resource use 19-3
 - degree of parallelism 18-5
 - description of 18-4
 - DS_MAX_QUERIES
 - parameter 37-21
 - DS_MAX_SCANS
 - parameter 37-22
 - DS_TOTAL_MEMORY
 - parameter 37-23
 - effect of table fragmentation 18-3
 - end user control of
 - resources 19-11
 - estimate shared memory 19-7
 - gates 18-20, 33-43
 - how used 18-14
 - limiting the priority 19-4
 - managing applications 19-9
 - maximum number of
 - queries 19-9
 - MAX_PDQPRIORITY
 - parameter 18-13, 37-43
 - mentioned 1-13
 - monitoring resources 33-40
 - monitoring resources allocated
 - for 33-40
 - OPTCOMPIND value 19-10
 - parameters used to control 18-11
 - PDQPRIORITY parameter 18-12, 37-43
 - principal components 18-4

- priority
 - effect of remote database 18-19
 - effect of stored procedure 19-6
 - setting dynamically 19-10
- queries that do not use PDQ 18-17
- queries that use PDQ 18-14
- resource allocation 18-11
- SET PDQPRIORITY
 - statement 18-12, 19-10
- statements affected by PDQ 18-18
- used with fragmentation 16-5
- uses of 18-3
 - when to use 18-8
- PDQPRIORITY environment variable
 - limiting PDQ priority 19-4
 - values of 18-12
 - with MGM 18-20
- PDQPRIORITY parameter
 - and MAX_PDQPRIORITY 37-43
 - effect of outer joins 18-19
- Pending transaction 39-104
- Performance
 - advantages of raw-disk management 1-12
 - and resident shared-memory 12-17
 - and shared memory 12-5
 - and yielding functions 10-13
 - effect of read-ahead 12-41
 - effects of VP controlled context switching 10-9
 - how frequently buffers are flushed 12-36
 - of CPU virtual processors 10-20
 - shared-memory connection 4-11, 4-12
- Performance configuration parameters
 - setting, using a text editor 13-11
 - setting, using ON-Monitor 13-12
- Performance tuning
 - and extent size 14-43
 - and foreground writes 12-51
 - and logical volume managers 14-49
 - blob space blobpage size 15-21
 - disk layout guidelines 14-36
 - logical-log size 22-5
 - LRU write 12-51
 - mechanisms 1-11
 - minimizing disk head movement 14-42
 - minimizing disk-head movement 14-42
 - moving the physical log 25-3
 - sample disk layout for optimal performance 14-45
 - spreading data across multiple disks 14-49
 - tuning amount of data logged 24-5
- Permissions, file 15-5
- PHYSBUFF parameter
 - and physical-log buffers 12-27
 - description of 37-56
 - mentioned 3-19
- PHYSDBS parameter
 - changing size and location 25-5
 - description of 37-57
 - mentioned 3-19
 - where located 24-8
- PHYSFILE parameter
 - changing size and location 25-5
 - description of 37-57
 - initial configuration value 24-6
 - mentioned 3-19
- Physical consistency, description of 26-5
- Physical log
 - and virtual processors 10-26
 - becoming full 24-7
 - before-image contents 24-5
 - buffer 24-9
 - changing size and location
 - possible methods 25-3
 - rationale 25-3
 - restrictions 25-4
 - using an editor 25-5
 - using ON-Monitor 25-4
 - using onparams 25-5
 - checking consistency 39-20
 - configuration parameters for 3-19
 - contiguous space 25-4
 - description of 24-3
 - effects of checkpoints on sizing 24-6
 - effects of frequent updating 24-6
 - ensuring does not become full 24-7
 - flushing of buffer 24-10
 - how emptied 24-11
 - in root dbspace 42-4
 - I/O, virtual processors 10-28
 - managing 25-3
 - monitoring 33-51
 - optimal storage of 14-38, 14-39
 - reinitialize shared memory 25-3
 - role in fast recovery 26-4, 26-5, 26-6
 - scenario for filling 24-8
 - size of 37-57
 - sizing guidelines 24-5
 - where located 24-8
- Physical logging
 - and archiving 24-4
 - and data buffer 24-10
 - and fast recovery 24-4
 - and simple large objects 24-5
 - description of 24-3
 - details of logging process 24-9
 - purpose of 24-4
 - which activity logged 24-4
- Physical page number 42-38
- Physical units of storage
 - description of 14-5
 - list of 14-3
- Physical-log buffer
 - amount written 12-49
 - and checkpoints 24-11
 - dbspace location 37-57
 - description of 12-27
 - events that prompt flushing 12-47
 - flushing of 12-46, 24-10
 - mentioned 12-48
 - monitoring 33-53
 - number of 12-27
 - PHYSBUFF parameter 12-27
 - role in dbspace logging 22-27, 24-10
 - size of 37-56
 - when it becomes full 12-48
- PIO virtual processors
 - description of 10-28
 - how many 10-28

- Planning for Universal Server
 - resources 3-4
- Point-in-time recovery
 - logical restore 22-16
- Point-in-time recovery. *See*
 - INFORMIX-Universal Server Backup and Restore Guide.
- Point-in-time recovery. *See also*
 - INFORMIX-Universal Server Archive and Backup Guide.
- Poll threads
 - and message queues 12-33
 - DBSERVERNAME
 - parameter 10-32
 - description of 10-33
 - how many 10-33
 - in NETTYPE parameter 37-47
 - multiple for a protocol 10-32
 - nettype entry 10-32
 - on CPU or network virtual processors 10-32
- Post-decision phase 34-6, 34-9
- Precommit phase 34-6
- Preparation, of database server environment 3-13
- Presumed-abort
 - optimization 34-10, 34-17
- Primary database server 29-4
- Primary key, use in fragmented table 42-40
- Printed manuals Intro-12
- Priorities for disk I/O 10-27
- Priority aging
 - description of 10-21
 - of CPU virtual processors 37-49
 - preventing 10-21
- Private environment file A-6
- Private installation
 - description of 3-9
- Private server
 - installing 3-9
- Privileges
 - on databases and tables 1-16
- Processes
 - compared to threads 10-5
 - DSA versus dual process architecture 10-9
 - that attach to shared-memory 12-11

- Processor affinity
 - AFF_NPROCS parameter 37-7
 - and AFF_SPROC parameter 37-8, 37-79
 - description of 10-11
 - set with VPCLASS 37-79
 - using 10-22
- Processor, locking for multiple or single 37-46
- Profile
 - displaying counts with onstat utility 39-79, 39-96
 - monitoring with SMI 38-21
 - setting counts to zero 39-80, 39-106
- Program counter and thread data 12-30
- Protocol
 - in NETTYPE parameter 37-47
 - specifying 10-32
- PSORT_DBTEMP environment variable
 - creating temporary implicit tables 14-30
 - relationship to
 - DBSPACETEMP 37-16
- PSORT_NPROCS environment variable, allocating sort memory 12-32

Q

- Queues
 - description of 10-16
 - disk I/O 10-30
 - ready 10-16
 - sleep 10-16
 - wait 10-18
- Quiescent mode
 - description of 7-4
 - with oninit utility 39-28

R

- RAID. *See* Redundant array of inexpensive disks.
- Raw device 14-6
 - and character-special interface 14-6
 - definition of 14-6
- Raw disk space 14-6
 - compared with cooked space 14-7
 - description of 14-6
 - how to allocate 15-5
 - in data storage 14-3
 - rationale for using 14-7
 - steps for allocating 14-7
- Raw-disk management 1-12
- RA_PAGES parameter
 - description of 37-58
 - purpose of 12-41
 - reading a page from disk 12-43
- RA_THRESHOLD parameter
 - description of 37-59
 - purpose of 12-41
- RDBMS. *See* Relational database management system.
- Read-ahead
 - description of 12-41
 - number of pages 37-58
 - RA_PAGES parameter 12-41
 - RA_THRESHOLD
 - parameter 12-41
 - threshold for 37-59
 - when it occurs 12-43
 - when used 12-41
- Read-only mode, description of 7-4
- Ready queue
 - description of 10-16
 - moving a thread to 10-17
- Reception buffer 29-9
- Recovery
 - by two-phase commit protocol 34-10
 - fast, description of 26-3
 - from media failure 27-5
 - parallel processing of 10-10
- Recovery mode, description of 7-4

- Recovery threads
 - off-line 37-52
 - on-line 37-52
- Redundant array of inexpensive disks (RAID), mirroring
 - alternative 27-7
- Referential constraints 20-5
- Regular buffers
 - and big buffers 12-30
 - description of 12-24
 - events that prompt flushing 12-46
 - how big 12-25
 - monitoring status of 12-25
- Relational database management system (RDBMS) 1-3
- Relay module, version 7.2
 - description of 4-53
 - example 4-53
 - example with three servers 4-55
- Release notes Intro-13
- Remainder page, description
 - of 42-41
- Remote client 4-21
- Remote database, effect on
 - PDQPRIORITY 18-19
- Remote hosts 4-21
- Replication server. *See* Data replication.
- Reserved pages
 - archive information 42-8
 - checking with oncheck 39-12, 39-20
 - checkpoint information 42-7
 - data-replication information 42-8
 - dbspace information 42-7
 - description of 42-6
 - estimating disk space for 14-35
 - location in root dbspace 42-4
 - optimal storage 14-38
 - organization in pairs 42-7
 - role in checkpoint
 - processing 42-9
 - validating with oncheck 31-4
 - viewing of contents 42-8
 - when updated 42-7
- RESIDENT parameter
 - description of 37-60
 - during initialization 9-10
 - interaction with onmode
 - utility 39-39
 - mentioned 13-15
- Resident portion of shared memory 9-7
- Resident shared memory 12-17
 - description of 12-17
 - internal tables 12-19
- RESIDENT parameter 37-60
 - setting configuration
 - parameters 13-7
 - turning on/off residency 39-39
- Resident shared-memory
 - conditions for 12-17
 - contents 12-17
- Resource allocation
 - effects of PDQ 18-11
- .rhosts file 4-21
- Roll back
 - in fast recovery 26-7
 - mentioned 1-14
- Roll forward
 - in fast recovery 26-7
 - mentioned 1-14
- Root dbspace
 - and temporary tables 14-21
 - calculating size of 14-33
 - description of 14-21
 - estimating disk space for 14-35
 - initial chunk 37-61
 - location of logical-log files 22-13
 - mentioned 3-15
 - mirroring 28-7, 37-45
 - specified by ROOTNAME
 - parameter 37-60
 - structure 42-4
 - using a link 37-61
- ROOTNAME parameter
 - description of 37-60
 - mentioned 3-15, 14-21
 - multiple residency 6-6
 - used by PHYSDBS 37-57
- ROOTOFFSET parameter
 - and multiple residency 5-4
 - description of 37-61
 - mentioned 3-16, 14-21
 - multiple residency 6-6
 - when is it needed 15-7
- ROOTPATH parameter
 - and multiple residency 5-4
 - description of 37-61
 - mentioned 3-15, 14-21
 - multiple residency 6-6
 - setting 3-14
 - specifying as a link 37-61
- ROOTSIZE parameter
 - description of 37-62
 - mentioned 3-16
- Round-robin distribution scheme
 - using 16-9
 - when to use 16-30
- Row
 - accommodating large rows 42-44
 - data row storage 42-40
 - displaying contents with
 - oncheck 39-21
 - effects of deletion on index 42-57
 - effects of modifying 42-44
 - linking of sections 42-42
 - maximum in a page 42-36
 - storage location 42-41
- Rowid
 - as component of index item 42-49
 - description of 42-37
 - effect of page compression 42-38
 - elements of 42-36
 - for fragmented table 16-20, 42-39
 - format 42-38
 - functions as forward
 - pointer 42-39
 - locking information derived
 - from 39-92
 - relation to slot table 42-36
 - stored in index pages 42-38
 - structure 42-38
 - where stored 42-38
- RSAM task control block 33-35

S

- Sample onconfig.std file A-14
- Sbpage
 - sizing recommendations 14-12
- Sbpage Structure 42-74
- Sbspace
 - adding a chunk 15-12, 15-18
 - chunk free extent list 42-72
 - chunk header structure 42-72
 - configuring Univesal Server,
and 3-28
 - creating using onspaces 15-16
 - default name 37-62
 - description of 1-7, 14-24
 - dropping 15-19
 - dropping using onspaces 15-20
 - metadata area structure 42-73
 - monitoring 33-67
 - multiple chunk sbpace 42-75
 - purpose of 14-24
 - sbpage structure 42-74
 - sbpace structure 42-71
 - sizing metadata 15-17
- SBSPACENAME parameter
 - mentioned 3-16
- Scalability
 - description of 10-3
- Scans
 - of indexes 12-41
 - of sequential tables 12-41
 - parallel processing of 10-10
- Scheduled system jobs 3-31
- Secondary database server 29-4
- Security
 - how enforced 1-16
 - isolating applications 5-3
 - of database server 1-16
 - risks with shared-memory
communications 4-12
- Security option, in options
field 4-40
- Segment identifier (shared-
memory) 12-13
- Segment. *See* Chunk.
- Semaphore, UNIX parameters
 - Shared memory
 - semaphore guidelines 13-6
- SERVERNUM parameter
 - and multiple residency 5-4, 5-5,
6-4, 6-5
 - and multiple Universal
Servers 12-13
 - description of 37-63
 - how used 12-12
 - mentioned 3-17
 - setting 3-14
- servicename
 - syntax rules 4-26
- servicename field in sqlhosts file
 - choosing an appropriate
name 4-36
 - description of 4-36
 - with IPX/SPX 4-38
 - with shared memory 4-36
 - with stream pipes 4-36
- servicename file A-12
- servicename.exp file A-11
- Service, in IPX/SPX 4-38
- Session
 - and active tblspace 12-23
 - and dictionary cache 12-31
 - and locks 12-22
 - and shared memory 12-30
 - and stored procedure cache 12-33
 - control block 10-12
 - description of 10-12
 - information in SMI tables 38-23,
38-24
 - monitoring 33-35
 - primary thread 12-30
 - shared-memory pool 12-29
 - sqlxec threads 10-6
 - threads 10-6
- Session control block 10-12
 - description of 12-30
 - shared memory 12-30
- SET DATASKIP statement 16-21,
16-28, 17-5
- SET EXPLAIN statement, use with
PDQ 19-9
- SET PDQRIORITY
 - statement 18-12, 19-10
 - limiting PDQ priority 19-4
 - use by MGM 18-21
- Share lock (buffer), description
of 12-35
- Shared data 12-5
- Shared library files A-6
- Shared memory
 - adding segment with
onmode 39-43
 - allocating 12-28
 - amount for sorting 12-32
 - and blobpages 12-61
 - and critical sections 12-55
 - and multiple Universal
Servers 12-15
 - and SERVERNUM
parameter 12-12
 - and SHMBASE parameter 12-12
 - attaching additional
segments 12-13, 12-15
 - attaching to 12-11
 - base address 37-65
 - buffer allocation 12-24
 - buffer locks 12-35
 - buffer pool 12-24, 24-9
 - buffer table 12-20
 - buffer-table hash table 12-20
 - buffer, frequency of
flushing 37-37
 - changing residency with
onmode 13-14, 39-39
 - changing with onmode 39-46
 - checkpoint 12-56
 - chunk table 12-20
 - communication 4-30
 - communications portion 12-33
 - configuring for multiple
residency 5-6
 - connection 4-11, 4-15
 - copying to a file 33-19
 - created during initialization 9-7
 - DataBlade modules,
considering 3-6
 - data-replication buffer 29-10
 - dbspace table 12-21
 - description of 12-5
 - dictionary cache 12-31
 - dumps 37-28, 37-29
 - dynamic management of 1-12
 - effect of UNIX kernel
parameters 13-4
 - estimate amount for PDQ 19-7
 - examining with SMI 38-5

- first segment 12-12
- for interprocess
 - communication 12-5
- global pool 12-33
- hash tables 12-19
- header 12-15, 12-19
- heaps 12-31
- how much 12-9
- how utilities attach 12-12
- how virtual processors
 - attach 12-12
- identifier 12-12
- initializing 9-3, 39-27
- initializing structures 9-7
- internal tables 12-19
- key value 12-12, 12-13
- largest allocation of resident
 - portion 12-24
- latches 12-34
- locating a page 12-43
- lock table 12-21
- logical-log buffer 12-26
- lower boundary address
 - problem 12-16
- mirror chunk table 12-20
- monitoring 33-19, 39-75
- mutexes 12-34
- operating system segments 12-9
- page-cleaner table 12-22
- performance 12-5
- performance options 13-12
- physical-log buffer 12-27, 37-56
- pools 12-28
- portions 12-7
- purposes of 12-5
- reinitializing 13-14
- resident portion 13-7
- resident portion, flag 37-60
- resident portion, mentioned 9-7
- saving copy of with onstat 39-79
- segment identifier 12-13
- segments, dynamically added,
 - size of 37-64
- session control block 12-30
- session data 12-30
- setting configuration
 - parameters 13-3
- SHMADD parameter 12-29
- SHMTOTAL parameter 12-9

- SHMVIRTSIZE parameter 12-29
- size displayed by onstat 12-9,
 - 39-81
- size of virtual portion 12-29
- sorting 12-32
- stacks 12-31
- STACKSIZE parameter 12-31
- stored procedures cache 12-33
- synchronizing buffer
 - flushing 12-45
- tables 12-19
- tblspace table 12-23
- the resident portion 12-17
- thread control block 12-30
- thread data 12-30
- thread isolation and buffer
 - locks 12-35
- total size 12-15
- transaction table 12-23
- Universal Server
 - requirements 12-12
 - use of SERVERNUM
 - parameter 37-63
 - user table 12-24
 - virtual portion 12-28, 12-29,
 - 13-10, 13-15
 - virtual segment, initial size 37-67
- Shared-memory buffer, maximum
 - number 37-9
- Shared-memory connection
 - example 4-47
 - how a client attaches 12-11
 - in nettype field 4-28
 - in servicename field 4-36
 - message 10-35
 - message buffers 12-33
 - virtual processor 10-31
- SHM virtual processor 10-31
- SHMADD parameter
 - description of 12-29
 - specifying value 37-64
- SHMBASE parameter
 - attaching first shared-memory
 - segment 12-13
 - description of 12-14, 37-65
 - warning 12-14

- shmem file
 - and assertion failures 31-7
 - and DUMPSHMEM
 - parameter 37-29
- shmem.xxx file A-11
- shmkey
 - attaching additional
 - segments 12-15
 - description of 12-13
- SHMTOTAL parameter
 - description of 12-9
 - specifying value 37-65
- SHMVIRTSIZE parameter
 - description of 37-67
 - specifying size of virtual shared
 - memory 12-29
- Shutdown
 - graceful 8-5
 - immediate 8-6
 - mode, description of 7-5
 - taking off-line 8-7
- Shutdown script
 - multiple residency 6-9
 - steps to perform 3-30
- Simple large object
 - absence of scanning or
 - compression 1-19
 - and physical logging 24-5
 - creating in a blobspace 12-63
 - effect of Committed Read
 - isolation 12-61
 - effect of Dirty Read
 - isolation 12-60
 - entering 1-19
 - how stored 42-63
 - how updated 42-65
 - illustration of blobspace
 - storage 12-63
 - in physical and logical log 14-23
 - modified by creating new simple
 - large object 42-65
 - monitoring for fullness 14-23
 - monitoring in a dbspace 33-74
 - page structure 42-65
 - role of simple-large-object
 - descriptor in
 - modifications 42-65
 - role of simple-large-object
 - descriptor in storage 42-64

- role of simple-large-object timestamps 12-59
- size limitations 42-65
- storage efficiency of 42-63
- storage on disk 42-64
- when created 12-62, 42-64
- when modified 42-65
- writing to a blob space 12-61
- Simple large object descriptor
 - description of 42-64
- Simple large objects 1-7
 - description of 1-7
- Simple-large-object descriptor
 - associated with simple-large object-timestamp 12-59
 - created during simple-large object-storage 12-62
 - description of 42-37
 - pointer becoming obsolete 12-60
 - structure and function 42-64
- Single processor computer 10-20
- SINGLE_CPU_VP parameter
 - and single processor computer 10-20, 10-21
 - description of 37-68
- Situations to avoid 32-3
- Sizing guidelines
 - logical log 22-7
 - physical log 24-5
- Skipping all unavailable fragments 16-23
- Skipping fragments
 - all fragments 16-23
 - effect on transactions 16-22
 - selected fragments 16-23
 - when to use feature 16-22
- Skipping selected fragments 16-23
- Sleep queues, description of 10-16
- Sleeping threads
 - forever 10-17
 - types of 10-16
- Slot table
 - description of 42-35
 - entry number 42-35
 - entry reflects changes in row size 42-38, 42-44
 - location on a db space page 42-34
 - relation to rowid 42-36
- slqhosts file
 - communications support module option 4-43
- Smart large object 1-7
 - and buffers 12-64
 - API, mentioned 1-7
 - default name 37-62
 - description of 1-7
 - fragmentation 16-9
 - monitoring 33-67
 - sb spaces, and 1-7
- SMI table
 - aborted table build 9-10
 - description of 38-5
 - during initialization 9-10
 - list of supported tables 38-8
 - monitoring buffer use 33-24
 - monitoring buffer-pool 33-27
 - monitoring checkpoints 33-18
 - monitoring chunks 33-64
 - monitoring data replication 33-78
 - monitoring databases 33-47
 - monitoring db spaces 33-58
 - monitoring fragmentation 33-67
 - monitoring latches 33-28
 - monitoring locks 33-31
 - monitoring log buffer use 33-54
 - monitoring logical-log files 33-51
 - monitoring sessions 33-39
 - monitoring shared memory 33-21
 - monitoring virtual processors 33-34
 - preparation during initialization 9-10
 - See also* sysmaster database.
 - See also* System monitoring interface.
- SOC virtual processors 10-31
- Sockets
 - connecting with 10-31
 - in nettype field 4-28
- Software dependencies Intro-4
- Sorting
 - and shared memory 12-32
 - as parallel process 10-10
- Split read 27-9
- SPSPACENAME parameter
 - description 37-62
- SPX virtual processors 10-31
- SQL statement
 - SET PDQPRIORITY 19-10
 - using temporary disk space 14-28
- SQLCA
 - warning flag when fragment skipped during query 37-13
- SQLEXEC environment variable,
 - example 4-54
- sqlxec thread
 - and client application 10-13
 - as user thread 10-6
 - role in client/server connection 10-34
- slqhosts file 3-11
 - and client redirection 29-24
 - and communications support module 4-8
 - dbservername field 4-27
 - defining multiple network addresses 10-37
 - description of 4-24
 - entries for multiple interface cards 10-38
 - example 4-24
 - for initialization 3-27
 - local loopback example 4-48
 - mentioned 1-11, A-12
 - multiple connection types,
 - example 4-45
 - multiple dbservernames 37-14
 - multiple residency 6-7
 - nettype field 4-27
 - network connection example 4-49
 - options field 4-38
 - servicename field 4-36
 - shared-memory example 4-47
 - specifying network poll threads 10-32
 - syntax rules 4-26
- SQLRM environment variable 4-55
- SQLRMDIR environment
 - variable 4-55
- Stack
 - and thread control block 10-15
 - description of 10-15
 - INFORMIXSTACKSIZE
 - environment variable 12-31
 - monitoring stack size 33-38
 - pointer 10-15

- size of 12-31
- STACKSIZE parameter 12-31
- thread 12-31
- STACKSIZE parameter
 - changing the stack size 12-31
 - description of 37-69
- STAGEBLOB parameter 37-70
- Standard database server 29-4
- Starting Universal Server
 - and initializing disk space 3-28
 - using oninit 39-27
- Startup script
 - multiple residency 6-9
 - multiple version of Universal Server 3-30
- Statement, SET
 - PDQPRIORITY 18-12
- Statistics. *See* onstat utility.
- status_vset_volnum.itgr file A-12
- Steps
 - for preparing a database server environment 3-13
 - for preparing multiple residence 6-4
- Stored procedure
 - effect of PDQ 18-18
 - effect on PDQPRIORITY 19-6
- Stored procedures cache 12-33
- stores7 database Intro-5
- Stream-pipe connection
 - advantages and disadvantages 4-13
 - in nettype field 4-28
 - in servicename field 4-36
- Structured Query Language 1-10
- Structured Query Language.
 - See also* SQL statement.
- Swapping memory 12-17
- Switching between threads 10-15
- Symbolic link
 - using with shared libraries A-6
- Symbolic link, using with
 - TAPEDEV 37-71
- Symmetric multiprocessing,
 - description of 10-4
- sysextspaces table 38-18
- sysfail.pidnum file A-13

- Sysmaster database
 - description 38-3
 - See also* System monitoring interface.
- sysmaster database
 - See also* SMI table.
 - buildsmi.xxx file A-5
 - description 38-3
 - failure to build A-5
 - functionality of 38-3
 - initialization 3-12
 - list of topics covered by 38-5
 - SMI tables 38-5
 - space required to build 38-4
 - types of tables 38-3
 - warning 38-4
 - when created 38-4
- sysmaster table
 - sysextspaces 38-18
- System catalog tables
 - and dictionary cache 12-31
 - and oncheck 39-17
 - disk space allocation for 42-76
 - how tracked 42-76
 - listing 39-13
 - location of 14-25
 - optimal storage of 14-39
 - sysfragments table 16-4, 42-21
 - tracking a new database 42-76
 - tracking a new table 42-78
 - validating with oncheck 31-5
- System failure, defined 26-4
- System monitoring interface (SMI)
 - See also* SMI table.
 - accessing SMI tables 38-6
 - and locking 38-7
 - and SPL 38-7
 - and triggers 38-7
 - description 38-3
 - SMI tables map 38-30
- tables
 - list of supported 38-8
 - sysadtinfo 38-9
 - sysaudit 38-10
 - syschkio 38-11
 - syschunks 38-11
 - sysconfig 38-13
 - sysdatabases 38-14
 - sysdbslocale 38-15

- sysdbspaces 38-15
- sysdri 38-17
- sysextents 38-18
- sysextspaces 38-18
- syslocks 38-19
- syslogs 38-20
- sysprofile 38-21
- sysptprof 38-23
- sysssesprof 38-24
- sysssessions 38-26
- syssseswts 38-28
- systabnames 38-29
- sysvpprof 38-30
- using SELECT statements 38-6
- using to monitor Universal Server 33-13
- using to obtain onstat information 38-33
- viewing tables with dbaccess 38-6
- System startup script, multiple residency 6-9
- System timer 10-16

T

Table

- creating, what happens on disk 42-76, 42-77
- deciding how to fragment 16-25
- description of 14-26
- disk-layout guidelines 14-39
- displaying allocation information 39-24
- high-use 14-42
- identifying its dbspace 42-21
- inheritance 1-4
- isolating 14-40
- migration *See Informix Migration Guide*.
- monitoring with SMI 38-29
- placing in a specific dbspace 14-27
- pseudotables 38-5
- purpose of 14-26
- recommendations for storage 14-42
- relationship to extent 14-26
- remote, used with PDQ 18-19

- SMI tables 38-5
- storage on middle partition of disk 14-42, 14-47
- temporary
 - cleanup during shared-memory initialization 14-29
 - effects of creating 42-80
 - estimating disk space for 14-34
 - message reporting cleanup 40-16
 - storage of explicit 14-29
 - storage of implicit 14-30
- Tape device
 - block size 37-39
- Tape management 3-31
- TAPEBLK parameter, mentioned 3-20
- TAPEDEV parameter
 - mentioned 3-20
 - setting 3-15
 - using a symbolic link 37-71
- TAPESIZE parameter, mentioned 3-20
- TBCONFIG environment variable A-9
- Tbldspace
 - description of 14-31
 - displaying information with onstat 39-80, 39-101
 - for fragmented index 16-20
 - for table fragment 16-19, 42-21
 - identifying its dbpspace 42-21
 - monitoring active tblspaces 33-31
 - monitoring with SMI 38-23
 - number 42-20
 - number displayed 39-101
 - number elements 42-21
 - purpose of 14-31
 - temporary tblspace during initialization 9-9
 - types of pages contained in 14-31
- Tbldspace number
 - components of 42-21
 - description of 42-20
 - displaying with onstat -t 39-101
 - for table fragment 42-21
 - includes dbpspace number 42-20
 - retrieving it from systables 42-20
- Tbldspace table
 - contents of 12-23
 - description of 12-23
- Tbldspace tblspace
 - bit-map page 42-22
 - description of 42-19
 - entries 42-19
 - location in a chunk 42-15
 - location in root dbpspace 42-4
 - size 42-22
 - structure and function 42-19
 - tracking new tables 42-78
- TCP/IP communication protocol
 - in hostname field 4-30
 - in nettype field 4-28
 - in servicename field 4-36
 - using 4-21
- TCP/IP connection
 - using a wildcard 4-32
 - using the internet IP address 4-31
 - using the TCP listen port number 4-37
- TCP/IP listen port number
 - finding the value 4-37
 - in servicename field 4-37
- tcldspace archive attributes
 - file A-13
- TEMP TABLE clause, of CREATE TABLE 16-7
- Template file for configuration 3-11
- Template for ONCONFIG file A-9
- Temporary dbpspace
 - advantages of 14-22
 - and data replication 29-36, 30-6
 - and DBSPACETEMP 14-29
 - and performance 14-22
 - described 14-21
- Temporary disk space
 - amount required for temporary tables 14-34
 - operations requiring 14-28
- Temporary table
 - and fragmentation 16-7
 - DBSPACETEMP parameter 37-16
 - during initialization 9-9
 - explicit 14-28
 - implicit 14-28
 - rules for use 37-16
 - where stored 14-29
 - with oninit utility 39-28
- TERM environment variable 3-12
- TERMCAP environment variable 3-12
- TERMINFO environment variable 3-12
- TEXT data type
 - Dirty Read isolation 12-60
 - mentioned 1-7
 - requires 4-bit bit map 42-26, 42-28
 - storage on disk 14-18
- Text editor
 - setting performance configuration parameters 13-13
 - setting shared memory parameters 13-8, 13-11
 - setting virtual processor parameters 11-6
- Thread
 - access to resources 10-9
 - accessing shared buffers 12-36
 - and heaps 12-31
 - and stacks 12-31
 - concurrency control 12-34
 - context of 10-13
 - control block 10-13, 12-30
 - description of 10-6
 - dynamic allocation of 1-13
 - for client applications 10-5
 - for primary session 10-13
 - for recovery 10-6
 - how virtual processors service 10-12
 - internal 10-6, 10-19
 - kernel asynchronous I/O 10-29
 - main_loop() 12-57
 - migrating 10-16
 - mirroring 10-6
 - monitoring 33-35
 - multiple concurrent 10-12
 - ON-Monitor 10-6
 - onstat information 39-80, 39-106
 - page cleaner 10-6, 12-45
 - relationship to a process 10-6
 - scheduling and synchronizing 10-12
 - session 10-6, 10-19
 - sleeping 10-17, 12-44

- supporting data replication 29-13
- switching between 10-15
- user 10-6
- waking up 10-17
- yielding 10-13
- Thread control block
 - creation of 12-30
 - role in context switching 10-14
- Thread-safe virtual processor 10-8
- Time intervals, configuration 3-25
- Time-out condition 39-88
- Timestamp
 - description of 12-59
 - location
 - blobpage blobpage 42-68, 42-69
 - dbspace page 42-34
 - dbspace simple-large-object page 42-66
 - mentioned 12-45
 - page-header and page-ending pair 12-59, 42-35
 - role in
 - data consistency 12-59
 - flushing physical-log buffer 12-49
 - simple-large-object pair 12-59
 - simple-large-object timestamps on a blobpage 42-69
- TLI. *See* Transport-layer interface.
- Transaction
 - factors which prevent closure 22-21
 - global
 - definition of 34-5
 - determining if implemented consistently 35-4
 - identification number, GTRID 35-8
 - tracking 34-29
 - heterogeneous commit 37-30
 - kill with onmode -Z 39-41
 - mixed result 34-25
 - monitoring 33-45
 - pending 39-104
 - piece of work, definition of 34-6
 - two-phase commit, examples 34-8

- Transaction logging
 - buffered 20-11
 - definition of 20-9
 - unbuffered 20-10
 - when to use 20-12
 - See also* Logging.
- Transaction table
 - description of 12-23
 - tracking with onstat 12-23
- Transport-layer interface
 - connecting with 10-31
 - in nettype field 4-28
 - virtual processors 10-31
- Trapping errors with onmode B-1
- Triggers, effect of PDQ 18-18
- Tuning
 - large number of users 37-48
 - use of NETTYPE parameter 37-47
- Two-phase commit
 - contrast with heterogeneous commit 34-38
- Two-phase commit protocol
 - automatic recovery 34-10
 - administrator's role 34-10
 - mechanisms for coordinator recovery 34-10
 - mechanisms for participant recovery 34-14
 - race condition 34-17
- configuration parameters for 34-37
- coordinator definition 34-5
- coordinator recovery mechanism 34-11
- description of 34-4, 34-5
- errors messages for 34-29
- global transaction definition 34-5
- global transaction identification number 35-8
- heuristic decisions
 - heuristic end-transaction 34-26
 - heuristic rollback 34-22
 - types of 34-21
- independent action
 - definition of 34-18
 - resulting in error condition 34-20
 - resulting in heuristic decisions 34-21

- results of 34-19
- what initiates 34-19
- logical-log records for 34-30
- messages 34-6
- participant recovery 34-14
- participant, definition of 34-6
- piece of work, definition of 34-6
- post-decision phase 34-6, 34-9
- precommit phase 34-6
- presumed-abort
 - optimization 34-10, 34-17
- requirements for flushing logical log records 34-32
- role of current server 34-5
- use of
 - DEADLOCK_TIMEOUT 34-37
 - use of TXTIMEOUT 34-37
- TXTIMEOUT parameter
 - and onmode -Z 34-28
 - description of 34-37, 37-73
 - in two-phase commit protocol 34-37
 - mentioned 3-25
 - role in automatic recovery 34-11, 34-15, 34-17
- Types of buffer writes 12-50

U

- Unbuffered logging
 - flushing the logical-log buffer 12-53
- Unbuffered transaction logging. *See* Logging.
- Unique ID 22-13
- Units of storage 14-3
- Universal Server
 - See* INFORMIX-Universal Server.
- Universal Server administrator
 - control of PDQ resources 19-11
 - role in managing PDQ 19-9
- UNIX devices
 - creating a link to a pathname 15-7
 - ownership, permissions on character-special 15-6
 - when are offsets needed 15-6

- UNIX files
 - ownership, permissions on
 - cooked files 15-5
 - using for data storage 14-8
- UNIX kernel parameters
 - description of 13-4
 - initial values 3-27
 - lower boundary address
 - parameter 12-16
- UNIX link command 15-7
- UNIX shutdown script 3-30
- UPDATE STATISTICS statement,
 - effect of PDQ 18-18
- Updates, disk 3-18
- Upgrading from INFORMIX-OnLine 3-8
- USEOSTIME parameter
 - description of 37-74
- User session
 - monitoring 33-35
 - monitoring with SMI 38-26
 - status codes 39-102
- User table
 - description of 12-24
 - maximum number of
 - entries 12-24
- User thread
 - acquiring a buffer 12-42
 - description of 10-6
 - in critical sections 12-55
 - monitoring 12-24
 - tracking 12-24
- User-defined access methods
 - mentioned 1-4
- User-defined data types
 - description of 1-6
 - mentioned 1-4
- User-defined routines
 - description of 1-8
 - ill-behaved 10-24
 - mentioned 1-4
 - on non-yielding virtual
 - processor 10-25
 - private installation, and 3-9
 - testing 3-9
 - virtual processors for 10-19
- User-defined virtual processor
 - starting in on-line mode 11-8

- User-defined virtual processors
 - purpose of 10-24
 - when to use 10-24
- Users, number of, in NETTYPE
 - parameter 37-47
- Utilities
 - attaching to shared-
 - memory 12-12
 - gcore 37-27, 37-28
 - oncheck 39-5
 - ondblog 39-25
 - oninit 39-27
 - onlog 39-30
 - onmode 39-35, 39-46
 - onparams 39-50
 - onstat 39-75
 - ontape 39-107
 - V option 39-3

V

- V option 39-3
- VARCHAR data type
 - byte locks 12-22
 - implications for data row
 - storage 42-40
 - indexing considerations 42-59
 - requires 4-bit bit map 42-26, 42-28
 - storage considerations 42-37
- Version, connecting to
 - different 4-53
- Virtual extension portion (shared
 - memory) 12-7
 - operating-system segments,
 - and 12-7
- Virtual portion (shared memory)
 - adding a segment 13-15
 - contents of 12-28, 12-29
 - global pool 12-33
 - setting configuration
 - parameters 13-10
 - size of 12-29
 - stacks 12-31
 - stored procedures cache 12-33
- Virtual processor
 - access to shared memory 12-7
 - adding and dropping 10-11
 - add / remove with onmode 39-44
- ADM class 10-16, 10-17
- ADT class 10-39
- advantages 10-8
- AIO class 10-29
- AIO, how many 10-30
- and context switching 10-9
- and ready queue 10-16
- as multithreaded process 10-6
- attaching to shared memory 12-7
- attaching to shared-
 - memory 12-12
- binding to CPUs 10-11
- classes of 10-7, 10-19
- coordination of access to
 - resources 10-9
- CPU class 10-19
- description of 10-5
- disk I/O 10-26
- dropping (CPU) in on-line
 - mode 11-10
- during initialization 9-8
- for DataBlade modules 10-19
- for user-defined routine 10-19
- how threads serviced 10-12
- LIO class 10-26
- LIO, how many 10-27
- logical-log I/O 10-27
- monitoring 33-32
- moving a thread 10-9
- MSC (miscellaneous) class 10-39
- network 10-31, 10-33
- number in AIO class 37-50
- number in CPU class 37-51
- OPT (optical) class 10-39
- parallel processing 10-10
- physical log I/O 10-28
- PIO class 10-26
- PIO, how many 10-28
- priority aging 37-49
- setting configuration
 - parameters 11-3
- sharing processing 10-8
- use of stack 10-15
- user-defined 10-24
- user-defined class 10-8
- user-defined, non-yielding 10-25
- VP class in NETTYPE
 - parameter 10-32, 37-47

VPCLASS parameter
 and AFF_NPROCS 37-7
 and AFF_SPROC 37-8
 and NOAGE 37-50
 and NUMAIOVPS 37-50
 configuring virtual
 processors 10-19
 default values 37-76
 description 37-75
 in ONCONFIG file 37-76
 mentioned 3-24, 10-8
 other parameters, and 3-24
 processor affinity 10-23
 reserved names 37-77
 set maximum VPs 37-79
 set number of VPs 37-79
 set processor affinity 37-79
 set with ON-Monitor 11-4
 used by
 DS_TOTAL_MEMORY 37-24
 user-defined classes 37-78
VP.servername.nnx file A-13

W

Wait queue
 and buffer locks 12-36
 description of 10-18
Waking up threads 10-17
Warning
 buildsmi script 38-5
 files on NIS systems 4-20
 interpreting after running
 oncheck -cc 31-5
 when fragment skipped during
 query processing 37-13
White space in ONCONFIG
 file 37-6
Wildcard addressing
 by a client application 4-34
 example 4-33
 in hostname field 4-32
WORM. *See* Optical devices.
Write types
 chunk write 12-51
 foreground write 12-50
 LRU write 12-51

X

X/Open compliance
 level Intro-14

Y

Yielding threads
 and ready queue 10-16
 at predetermined point 10-13
 description of 10-13
 on some condition 10-13
 switching between 10-12
ypcat hosts, UNIX command 4-20
ypcat services, UNIX
 command 4-20
ypmatch command 4-37

