

SIEMENS
NIXDORF

INFORMIX

INFORMIX-OnLine Dynamic Server V6.0

(UNIX)
Administrator's Guide
Volume 1

Uns interessiert Ihre Meinung
zu dieser Druckschrift.

Schicken Sie uns bitte eine Kopie dieser Seite,
wenn Sie uns Hinweise geben wollen:

- zum Inhalt
- zur Form
- zum Produkt.

Dafür bedanken wir uns im voraus.
Mit freundlichen Grüßen,
Ihre

Siemens Nixdorf Informationssysteme AG
Unternehmenskommunikation
81730 München

Fax: (0 89) 6 36-4 97 68

We would like to know
your opinion on this publication.

Please send us a copy of this page
if you have any criticism on:

- the contents
- the layout
- the product.

We would like to thank you in advance
for your comments.
With kind regards,

Siemens Nixdorf Informationssysteme AG
Corporate Communication
D-81730 München

Fax: (00 49) 89 6 36-4 97 68

Ihre Meinung/Your opinion:

Bestellnummer dieser Druckschrift:

Order number of this manual:
U9636-J-Z265-2-7600

INFORMIX-OnLine Dynamic Server V6.0

(UNIX)

Administrator's Guide

Volume 1

Edition April 1994

THE INFORMIX SOFTWARE AND USER MANUAL ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE INFORMIX SOFTWARE AND USER MANUAL IS WITH YOU. SHOULD THE INFORMIX SOFTWARE AND USER MANUAL PROVE DEFECTIVE, YOU (AND NOT INFORMIX OR ANY AUTHORIZED REPRESENTATIVE OF INFORMIX) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR, OR CORRECTION. IN NO EVENT WILL INFORMIX BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF OR INABILITY TO USE SUCH INFORMIX SOFTWARE OR USER MANUAL, EVEN IF INFORMIX OR AN AUTHORIZED REPRESENTATIVE OF INFORMIX HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY. IN ADDITION, INFORMIX SHALL NOT BE LIABLE FOR ANY CLAIM ARISING OUT OF THE USE OF OR INABILITY TO USE SUCH INFORMIX SOFTWARE OR USER MANUAL BASED UPON STRICT LIABILITY OR INFORMIX'S NEGLIGENCE. SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS AND YOU MAY ALSO HAVE OTHER RIGHTS, WHICH VARY FROM STATE TO STATE.

All rights reserved. No part of this work covered by the copyright hereon may be reproduced or used in any form or by any means – graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems – without permission of the publisher.

Published by INFORMIX® Press

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

INFORMIX® and C-ISAM® are registered trademarks of Informix Software, Inc.

UNIX® is a registered trademark, licensed exclusively by the X/Open Company Ltd. in the United Kingdom and other countries.

X/Open® is a registered trademark of X/Open Company Ltd. in the United Kingdom and other countries.

MS® is a trademark of Microsoft Corporation.

PostScript is a registered trademark of Adobe Systems Incorporated.

Some of the products or services mentioned in this document are provided by companies other than Informix. These products or services are identified by the trademark or servicemark of the appropriate company. If you have a question about one of those products or services, please call the company in question directly.

Documentation Team: Bob Berry, Sally Cox, Tom DeMott, Jenny Robertson, Judith Sherwood, Rob Weinberg, Chris Willis, Eileen Wollam.

RESTRICTED RIGHTS LEGEND

The Informix software and accompanying materials are provided with Restricted Rights. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (c) (1) (ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or subparagraphs (c) (1) and (2) of the Commercial Computer Software-Restricted Rights at 48 CFR 52.227-19, as applicable (and any other applicable license provisions set forth in the Government contract).

Copyright © 1981–1994 by Informix Software, Inc.

Preface

The *INFORMIX-OnLine Dynamic Server Administrator's Guide* is a complete guide to the features that make up the **INFORMIX-OnLine Dynamic Server** relational database server.

You should have some familiarity with relational database concepts to use this manual. However, a knowledge of Structured Query Language (SQL) would be useful. Informix SQL is described in detail in a separate set of manuals called the *Informix Guide to SQL: Tutorial* and the *Informix Guide to SQL: Reference*.

The *INFORMIX-OnLine Dynamic Server Administrator's Guide* is both a user guide and a reference manual. The first nine sections cover important basic information about the product. The last section contains reference material for using **INFORMIX-OnLine Dynamic Server**.

Summary of Chapters

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

- This Preface provides general information about the manual and lists additional reference materials that will help you understand **INFORMIX-OnLine Dynamic Server** concepts.
- The Introduction tells how **INFORMIX-OnLine Dynamic Server** fits into the Informix family of products and manuals, explains how to use the manual, introduces the demonstration database from which the product examples are drawn, describes the **Informix Messages and Corrections** product, and lists the new features for Version 6.0 of Informix database server products.
- The section "What is INFORMIX-OnLine?" 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 15.
- The section “Logging and Log Administration” is made up of Chapters 16 through 22.
- The section “Fault Tolerance” is made up of Chapters 23 through 28.
- The section “Monitoring and Performance” is made up of Chapters 29 and 30.
- The section “Data Migration” is made up of Chapter 31.
- The section “Distributed Data” is made up of Chapters 32 and 33.
- The section “Reference” is made up of Chapters 34 through 42.
- The Index includes references throughout the *INFORMIX-OnLine Dynamic Server Administrator’s Guide*.

Related Reading

If you want additional technical information on database management, consult the following texts by C. J. Date:

- *An Introduction to Database Systems, Volume I* (Addison-Wesley Publishing, 1990)
- *An Introduction to Database Systems, Volume II* (Addison-Wesley Publishing, 1983)

This guide assumes that you are familiar with your computer operating system. If you have limited UNIX system experience, you might want to look at your operating system manual or a good introductory text before you read this manual.

Some suggested texts about UNIX systems follow:

- *A Practical Guide to the UNIX System*, Second Edition, by M. Sobell (Benjamin/Cummings Publishing, 1989)
- *A Practical Guide to UNIX System V* by M. Sobell (Benjamin/Cummings Publishing, 1985)
- *Introducing the UNIX System* by H. McGilton and R. Morgan (McGraw-Hill Book Company, 1983)
- *UNIX for People* by P. Birns, P. Brown, and J. Muster (Prentice-Hall, 1985)

List of Chapters

What Is INFORMIX-OnLine Dynamic Server?

- Chapter 1 What Is INFORMIX-OnLine Dynamic Server?
Chapter 2 Overview of OnLine Administration

Configuration

- Chapter 3 Installing and Configuring OnLine
Chapter 4 Configuring Connectivity
Chapter 5 What Is Multiple Residency?
Chapter 6 Using Multiple Residency

Modes and Initialization

- Chapter 7 What Are OnLine Operating Modes?
Chapter 8 Managing Modes
Chapter 9 What Is Initialization?

Disk, Memory, and Process Management

- Chapter 10 Where Is Data Stored?
Chapter 11 Managing Disk Space
Chapter 12 What Is the Dynamic Scalable Architecture?

-
- Chapter 13** **Managing Virtual Processors**
 - Chapter 14** **OnLine Shared Memory**
 - Chapter 15** **Managing OnLine Shared Memory**

Logging and Log Administration

- Chapter 16** **What Is Logging?**
- Chapter 17** **Managing Database Logging Status**
- Chapter 18** **What Is the Logical Log?**
- Chapter 19** **Managing Logical-Log Files**
- Chapter 20** **What Is Physical Logging?**
- Chapter 21** **Managing the Physical Log**
- Chapter 22** **What Is Fast Recovery?**

Fault Tolerance

- Chapter 23** **What Is Mirroring?**
- Chapter 24** **Using Mirroring**
- Chapter 25** **What Is Data Replication?**
- Chapter 26** **Using Data Replication**
- Chapter 27** **What Is Consistency Checking?**
- Chapter 28** **Situations to Avoid**

Monitoring and Performance

- Chapter 29** **Monitoring OnLine**
- Chapter 30** **Improving Performance**

Data Migration

Chapter 31 **Data Migration**

Distributed Data

Chapter 32 **What Is Two-Phase Commit?**

Chapter 33 **How to Manually Recover from Failed Two-Phase Commit Protocol**

Reference

Chapter 34 **ON-Monitor**

Chapter 35 **OnLine Configuration Parameters**

Chapter 36 **The sysmaster Database**

Chapter 37 **OnLine Utilities**

Chapter 38 **OnLine Message Log Messages**

Chapter 39 **Interpreting Logical-Log Records**

Chapter 40 **OnLine Disk Structure and Storage**

Chapter 41 **OnLine Environment Variables**

Chapter 42 **Files Used by OnLine**

Table of Contents

Introduction

INFORMIX-OnLine Dynamic Server and Other Informix Products	3
Other Useful Documentation	4
How to Use This Manual	5
Typographical Conventions	5
Command-Line Conventions	5
Example Code Conventions	8
Useful On-Line Files	8
ASCII and PostScript Error Message Files	9
The Demonstration Database	9
Creating the Demonstration Database on INFORMIX-OnLine Dynamic Server	10
New Features in INFORMIX-OnLine Dynamic Server Version 6.0	11

What Is INFORMIX-OnLine Dynamic Server?

Chapter 1

What Is INFORMIX-OnLine Dynamic Server?

Chapter Overview	1-3
What Is OnLine?	1-3
Client/Server Architecture	1-4
High Performance	1-5
Fault Tolerance and High Availability	1-6
Multimedia Support	1-8
Distributed Data Queries	1-8
Database Server Security	1-8
Who Uses OnLine?	1-9
End Users	1-9
Application Developers	1-9
Database Administrators	1-10

OnLine Administrators	1-10
OnLine Operators	1-10
Features Beyond the Scope of OnLine	1-10
No Bad-Sector Mapping	1-10
No Blob Scanning or Compression	1-11

Chapter 2

Overview of OnLine Administration

Chapter Overview	2-3
Initial Tasks	2-3
Routine Tasks	2-3
Changing Modes	2-4
Archiving Data and Backing Up Logical-Log Files	2-4
Monitoring OnLine Activity	2-4
Checking for Consistency	2-4
Configuration Tasks	2-4
Managing OnLine Instances	2-5
Managing Database Logging Status	2-5
Logical-Log Administration	2-5
Physical-Log Administration	2-5
Using Auditing	2-6
Using Mirroring	2-6
Using Data Replication	2-6
Managing Shared Memory	2-6
Managing Virtual Processors	2-7

Configuration

Chapter 3

Installing and Configuring OnLine

Chapter Overview	3-3
Planning for INFORMIX-OnLine Dynamic Server	3-4
Consider Your Priorities	3-4
Consider Your Resources	3-4
Administering OnLine	3-5
Installing INFORMIX-OnLine Dynamic Server	3-6
Installing OnLine When No Other Informix Products Are Present	3-6
Installing OnLine When Other Informix Products Are Present	3-6
Installing OnLine When SE Is Already Present	3-6
Upgrading an Earlier Version of OnLine	3-7
Configuration Overview	3-8
Configuration Files	3-8
Environment Variables Used by OnLine	3-9
Multiple OnLine Database Servers	3-10

Configuring a Learning Environment	3-10
Log in as User informix	3-11
Choose Names	3-11
Set Environment Variables	3-11
Allocate Disk Space for Data Storage	3-12
Prepare the ONCONFIG Configuration File	3-13
Prepare the Connectivity File	3-15
Start OnLine Running	3-16
Practice Using OnLine	3-17
Configuring a Production Environment	3-17
Set Environment Variables	3-18
Prepare the ONCONFIG Configuration File	3-18
Overview of Configuration Parameters	3-19
Allocate Disk Space	3-26
Prepare the Connectivity File	3-26
Prepare the ON-Archive Configuration File	3-26
Prepare for Native Language Support	3-27
Evaluate UNIX Kernel Parameters	3-27
Start OnLine and Initialize Disk Space	3-27
Create Blobspaces and Dbspaces	3-28
Do Administrative Tasks	3-28

Chapter 4

Configuring Connectivity

Chapter Overview	4-3
Types of Client/Server Connections	4-3
Shared-Memory Connections	4-4
Network Connections	4-5
Connectivity Files	4-7
Network Configuration Files	4-7
Network Security Files	4-9
The \$INFORMIXDIR/etc/sqlhosts File	4-10
ONCONFIG Parameters for Connectivity	4-17
The DBSERVERNAME Configuration Parameter	4-17
The DBSERVERALIASES Configuration Parameter	4-18
Environment Variables for Network Connections	4-19
Examples of Client/Server Configurations	4-19
Using a Shared-Memory Connection	4-20
Using a Local Loopback Connection	4-21
Using a Network Connection	4-21
Using Multiple Connection Types	4-23
Accessing Multiple 6.0 OnLine Database Servers	4-25

Using the 6.0 Relay Module	4-26
Using 5.0 INFORMIX-STAR or 5.0 INFORMIX-NET	4-29
Using a 6.0 Client Application with a 5.0 Database Server	4-30

Chapter 5

What Is Multiple Residency?

Chapter Overview	5-3
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	5-4

Chapter 6

Using Multiple Residency

Chapter Overview	6-3
Planning for Multiple Residency	6-3
Preparing for Multiple Residency	6-4
Prepare a Configuration File	6-4
Set Your ONCONFIG Environment Variable	6-4
Edit the New Configuration File	6-5
Add Connection Information	6-6
Update the \$INFORMIXDIR/etc/sqlhosts File	6-6
Initialize Disk Space	6-6
Prepare Archive and Backup Environment	6-7
Update the Operating System Boot File	6-8
Check Users' INFORMIXSERVER Environment Variables	6-8

Modes and Initialization

Chapter 7

What Are OnLine Operating Modes?

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

Chapter 8

Managing Modes

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

Chapter 9

What Is Initialization?

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

Disk, Memory, and Process Management

Chapter 10

Where Is Data Stored?

Chapter Overview	10-3
Overview of Data Storage	10-3
What Are the Physical Units of Storage?	10-4
What Is a Chunk?	10-4

What Is a Page?	10-9
What Is A Blobpage?	10-10
What Is an Extent?	10-11
What Are the Logical Units of Storage?	10-14
What Is a Dbspace?	10-15
What Is a Blobspace?	10-19
What Is a Database?	10-20
What Is a Table?	10-21
What Is a Tblspace?	10-25
How Much Disk Space Do You Need to Store Your Data?	10-27
Estimate the Size of the Root Dbspace	10-27
Estimate Space Required by Databases Including Overhead and Growth	10-30
Disk-Layout Guidelines	10-30
Strive to Associate Partitions with Chunks	10-31
Consider Mirroring	10-31
Isolate High-Use Tables	10-31
Group Your Tables with Archive and Restore in Mind	10-32
Spread a Single Table Across Multiple Disk Devices to Reduce Contention	10-33
Place High-Use Tables on Middle Partition of Disk	10-33
Spread Your Temporary Storage Space Across Multiple Disks	10-34
Optimize Table Extent Sizes	10-34
Move the Logical and Physical Logs from the Root Dbspace	10-35
Take into Account Archive and Restore Performance	10-36
Sample Disk Layouts	10-37
What Is a Logical Volume Manager?	10-44

Chapter 11

Managing Disk Space

Chapter Overview	11-3
Allocating Disk Space	11-3
Allocating Cooked File Space	11-4
Allocating Raw Disk Space	11-5
Initializing Disk Space	11-7
Initializing Disk Space with ON-Monitor	11-7
Initializing Disk Space with oninit	11-7
Creating a Dbspace	11-8
Creating a Dbspace Using ON-Monitor	11-9
Creating a Dbspace Using onspaces	11-9
Adding a Chunk to a Dbspace	11-10
Adding a Chunk	11-10
Creating a Blobspace	11-12
Determining OnLine Page Size	11-13

Creating a BlobSpace Using ON-Monitor	11-13
Creating a BlobSpace Using onspaces	11-14
Adding a Chunk to a BlobSpace	11-14
Dropping a Chunk from a DbSpace Using onspaces	11-14
Dropping a Chunk from a BlobSpace	11-15
Dropping a DbSpace or BlobSpace	11-15
Dropping a DbSpace or BlobSpace Using ON-Monitor	11-16
Dropping a DbSpace or BlobSpaces Using onspaces	11-16
Optimizing BlobSpace Blobpage Size	11-16
Determining BlobSpace Storage Efficiency	11-17
BlobSpace Storage Statistics	11-17
Determining Blobpage Fullness with oncheck -pB	11-17
Managing Extents	11-19
Managing Tables	11-21
Reclaiming Space in an Empty Extent Using Alter Index	11-21
Reclaiming Space in an Empty Extent Using the UNLOAD and LOAD Statements	11-21

Chapter 12

What Is the Dynamic Scalable Architecture?

Chapter Overview	12-3
What Is a Virtual Processor?	12-4
What Is a Thread?	12-5
Types of Virtual Processors	12-5
Advantages of Virtual Processors	12-6
How Virtual Processors Service Threads	12-9
Control Structures	12-10
Context Switching	12-10
Stacks	12-12
Queues	12-14
Mutexes	12-16
Virtual Processor Classes	12-16
CPU Virtual Processors	12-16
Disk I/O Virtual Processors	12-20
Network Virtual Processors	12-24
Administration Virtual Processors	12-30
Optical Virtual Processor	12-30
Audit Virtual Processor	12-31

Chapter 13

Managing Virtual Processors

Chapter Overview	13-3
Setting Virtual Processor Configuration Parameters	13-3
Setting Virtual Processor Configuration Parameters Using ON-Monitor	13-3

Setting Virtual Processor Configuration Parameters Using a Text Editor	13-5
Starting and Stopping Virtual Processors	13-6
Adding Virtual Processors in On-Line Mode	13-7
Dropping CPU Virtual Processors in On-Line Mode	13-9

Chapter 14

OnLine Shared Memory

Chapter Overview	14-5
What Is Shared Memory?	14-5
How OnLine Uses Shared Memory	14-6
How OnLine Allocates Shared Memory	14-8
How Much Shared Memory Does OnLine Use?	14-10
What Processes Attach to OnLine Shared Memory?	14-10
How a Client Attaches to the Communications Portion	14-10
How Utilities Attach to Shared Memory	14-11
How Virtual Processors Attach to Shared Memory	14-11
The Resident Portion of OnLine Shared Memory	14-15
Shared-Memory Header	14-17
Shared-Memory Internal Tables	14-18
Shared-Memory Buffer Pool	14-23
The Virtual Portion of OnLine Shared Memory	14-25
How OnLine Manages the Virtual Portion of Shared Memory	14-26
What Is in the Virtual Portion of Shared Memory	14-26
The Communications Portion of OnLine Shared Memory	14-29
Concurrency Control	14-30
Shared-Memory Mutexes	14-30
Shared-Memory Buffer Locks	14-31
How OnLine Threads Access Shared Buffers	14-32
OnLine LRU Queues	14-32
Read Ahead	14-35
How an OnLine Thread Accesses a Buffer Page	14-36
How OnLine Flushes Data to Disk	14-39
Flushing the Physical-Log Buffer	14-39
How OnLine Synchronizes Buffer Flushing	14-41
Types of Writes that Prompt Flushing Activity	14-42
Flushing the Logical-Log Buffer	14-44
How OnLine Achieves Data Consistency	14-46
Critical Sections	14-46
OnLine Checkpoints	14-47
OnLine Timestamps	14-50
Writing Data to a BlobSpace	14-52

Chapter 15	Managing OnLine Shared Memory
	Chapter Overview 15-3
	Setting Shared-Memory Configuration Parameters 15-3
	UNIX Kernel Configuration Parameters 15-3
	OnLine Shared-Memory Configuration Parameters 15-6
	Reinitializing Shared Memory 15-14
	Turning on or Turning off Residency for Resident Shared Memory 15-15
	Turning on or Turning off Residency While OnLine is in On-Line Mode 15-15
	Turning on or Turning off Residency for the Future 15-15
	Adding a Segment to the Virtual Portion of Shared Memory 15-16
	Forcing a Checkpoint 15-16

Logging and Log Administration

Chapter 16	What Is Logging?
	Chapter Overview 16-3
	Which OnLine Processes Require Logging? 16-3
	What OnLine Activity Is Logged? 16-5
	Activity That Is Always Logged 16-6
	Activity Logged for Databases with Transaction Logging 16-6
	Are Blobs Logged? 16-7
	What Is Transaction Logging? 16-7
	The Database Logging Status 16-8
	When to Use or not Use Transaction Logging 16-9
	When to Buffer or not Buffer Transaction Logging 16-10
	Who Can Set or Change Logging Status 16-10

Chapter 17	Managing Database Logging Status
	Chapter Overview 17-3
	About Changing Logging Status 17-3
	Modifying Database Logging Status Using ON-Archive 17-5
	Turning on Transaction Logging Using ON-Archive 17-5
	Ending Logging Using ON-Archive 17-6
	Changing Buffering Status Using ON-Archive 17-6
	Making a Database ANSI-Compliant Using ON-Archive 17-6
	Modifying Database Logging Status Using ontape 17-6
	Turning on Transaction Logging Using ontape 17-7
	Ending Logging Using ontape 17-7
	Changing Buffering Status Using ontape 17-7
	Making a Database ANSI-Compliant Using ontape 17-8
	Modifying Database Logging Status Using ON-Monitor 17-8

Chapter 18

What Is the Logical Log?

- Chapter Overview 18-3
- What Is the Logical Log? 18-3
- What Is a Logical-Log File? 18-4
- How Big Should the Logical Log Be? 18-5
 - Performance Considerations 18-5
 - Long-Transaction Consideration 18-6
 - Logical-Log Size Guidelines 18-6
 - Determining the Size of the Logical Log 18-7
- What Should Be the Size and Number of Logical-Log Files? 18-7
- Where Should Logical-Log Files Be Located? 18-8
- How Are Logical-Log Files Identified? 18-8
- What Are the Status Flags of Logical-Log Files? 18-9
- Why Do Logical-Log Files Need to Be Backed Up? 18-10
- When Are Logical-Log Files Freed? 18-11
 - When Does OnLine Attempt to Free a Log File? 18-11
 - What Happens If the Next Logical-Log File Is Not Free? 18-11
 - Avoiding Long Transactions 18-12
- What Are the Logical-Log Administration Tasks Required for Blobspaces? 18-15
 - Switching Logical-Log Files to Activate Blobspaces 18-15
 - Switching Logical-Log Files to Activate New BlobSpace Chunks 18-16
 - Backing Up Logical-Log Files to Free Blobpages 18-16
- What Is the Logging Process? 18-18
 - DbSpace Logging 18-18
 - BlobSpace Logging 18-20

Chapter 19

Managing Logical-Log Files

- Chapter Overview 19-3
- Adding a Logical-Log File 19-3
 - Adding a Log File Using ON-Monitor 19-4
 - Adding a Log File Using **onparams** 19-4
 - Adding a Log File with a New Size 19-5
- Dropping a Logical-Log File 19-5
 - Dropping a Logical-Log File Using ON-Monitor 19-6
 - Dropping a Logical-Log File Using **onparams** 19-6
- Moving a Logical-Log File to Another DbSpace 19-6
 - An Example of Moving Logical-Log Files 19-7
- Changing the Size of Logical-Log Files 19-7
- Changing Logical-Log Configuration Parameters 19-8
 - Changing LOGSIZE or LOGFILES 19-8
 - Changing LOGSMAX, LTXHWM, or LTXEHWM 19-9
- Freeing a Logical-Log File 19-10

	Freeing a Log File with Status A	19-10
	Freeing a Log File with Status U	19-11
	Freeing a Log File with Status U-B	19-11
	Freeing a Log File with Status U-C or U-C-L	19-11
	Freeing a Log File with Status U-B-L	19-12
	Switching to the Next Logical-Log File	19-12
Chapter 20	What Is Physical Logging?	
	Chapter Overview	20-3
	What Is Physical Logging?	20-3
	What Is the Purpose of the Physical Logging?	20-3
	What OnLine Activity Is Physically Logged?	20-4
	What Is the Physical Log?	20-5
	How Big Should the Physical Log Be?	20-5
	Where Is the Physical Log Located?	20-7
	Details of Physical Logging	20-8
	Page Is Read into the Shared-Memory Buffer Pool	20-8
	A Copy of the Page Buffer Is Stored in the Physical-Log Buffer	20-8
	Change Is Reflected in the Data Buffer	20-9
	Physical-Log Buffer Is Flushed to the Physical Log	20-9
	Page Buffer Is Flushed	20-9
	When Checkpoint Occurs, Physical-Log Buffer Is Flushed and Physical Log Is Emptied	20-9
	How the Physical Log Is Emptied	20-9
Chapter 21	Managing the Physical Log	
	Chapter Overview	21-3
	Changing the Physical-Log Location and Size	21-3
	Why Change Physical-Log Location and Size?	21-3
	Before You Make the Changes	21-4
	Using ON-Monitor to Changing Physical-Log Location or Size	21-4
	Using an Editor to change Physical-Log Location and Size	21-5
	Using onparams to Change Physical-Log Location or Size	21-5
Chapter 22	What Is Fast Recovery?	
	Chapter Overview	22-3
	What Is Fast Recovery?	22-3
	When Is Fast Recovery Needed?	22-3
	When Does OnLine Initiate Fast Recovery?	22-4
	Fast Recovery and Buffered Logging	22-4
	Fast Recovery and No Logging	22-4
	Details of Fast Recovery	22-5
	Return to the Last-Checkpoint State	22-5

Find the Checkpoint Record in the Logical Log 22-6
Roll Forward Logical Log Records 22-7
Roll Back Incomplete Transactions 22-8

Fault Tolerance

Chapter 23

What Is Mirroring?

Chapter Overview 23-3
What Is Mirroring? 23-3
 What Are the Benefits of Mirroring? 23-4
 What Are the Costs of Mirroring? 23-4
 What Happens If You Do Not Mirror? 23-5
 What Should You Mirror? 23-5
 What Mirroring Alternatives Exist? 23-5
The Mirroring Process 23-6
 What Happens When You Create a Mirror Chunk? 23-6
 What Are Mirror Status Flags? 23-7
 What Is Recovery? 23-7
 What Happens During Processing? 23-8
 What Happens If You Stop Mirroring? 23-10
 What Is the Structure of a Mirror Chunk? 23-10

Chapter 24

Using Mirroring

Chapter Overview 24-3
Steps Required for Mirroring Data 24-3
Enabling Mirroring 24-4
Allocating Disk Space for Mirrored Data 24-5
Starting Mirroring 24-5
 Mirroring the Root Dbspace During Initialization 24-6
 Starting Mirroring for Unmirrored Dbspaces 24-6
 Starting Mirroring for New Dbspaces 24-7
Adding Mirror Chunks 24-8
Changing the Mirror Status 24-8
 Taking Down a Mirror Chunk 24-8
 Recovering a Mirrored Chunk 24-9
Relinking a Chunk to a Device After a Disk Failure 24-10
Ending Mirroring 24-10

Chapter 25

What Is Data Replication?

Chapter Overview 25-3
What Is Data Replication? 25-3
 What Is OnLine High Availability Data Replication? 25-4

How Does Data Replication Work?	25-8
How Is the Data Initially Replicated?	25-8
How Are Updates to the Primary Reproduced on the Secondary?	25-9
When Are Log Records Sent?	25-10
What Threads Handle Data Replication?	25-13
Checkpoints Between Database Servers	25-13
How Is Data Synchronization Tracked?	25-14
Data-Replication Failures	25-14
What Are Data-Replication Failures?	25-14
How Are Data-Replication Failures Detected?	25-15
What Happens When a Data-Replication Failure is Detected?	25-15
Administrative Considerations After Data-Replication Failure	25-16
Redirection and Connectivity for Data-Replication Clients	25-19
Designing Clients for Redirection	25-20
Automatic Redirection: Using DBPATH	25-20
Administrator-Controlled Redirection: Changing the <code>sqlhosts</code> File	25-22
User-Controlled Redirection: <code>INFORMIXSERVER</code>	25-25
Handling Redirection Within an Application	25-26
Comparison of Different Redirection Mechanisms	25-28
Designing Clients to Use the Secondary Database Server	25-29
No Data Modification Statements	25-29
Locking and Isolation Level	25-30
Using Temporary Dbspaces for Sorting and Temporary Tables	25-31

Chapter 26

Using Data Replication

Chapter Overview	26-3
Planning for Data Replication	26-3
Configuring Data Replication	26-4
Meeting Hardware and Operating-System Requirements	26-4
Meeting Database and Data Requirements	26-5
Meeting Database Server Configuration Requirements	26-5
Configuring Data-Replication Connectivity	26-8
Starting Data Replication for the First Time	26-9
Performing Basic OnLine Administration Tasks	26-12
Changing Database Server Configuration Parameters	26-12
Archiving and Logical-Log File Backups	26-12
Changing the Logging Status of Databases	26-13
Adding and Dropping Chunks, Dbspaces, and Blobspaces	26-13
Using and Changing Mirroring of Chunks	26-13
Managing the Physical Log	26-14
Managing the Logical Log	26-14
Managing Virtual Processors	26-15

Managing Shared Memory	26-15
Changing the Database Server Mode	26-15
Changing the Database Server Type	26-16
Restoring Data If Media Failure Occurs	26-18
Restarting Data Replication After a Failure	26-20
Restarting After Critical Data Is Damaged	26-20
Restarting If Critical Data Is Not Damaged	26-23

Chapter 27

What Is Consistency Checking?

Chapter Overview	27-3
Performing Periodic Consistency Checking	27-3
Verify Consistency	27-4
Monitor for Data Inconsistency	27-6
Retain Consistent Level-0 Archive	27-7
Dealing with Corruption	27-7
Symptoms of Corruption	27-8
Run oncheck First	27-8
I/O Errors on a Chunk	27-8
Collecting Diagnostic Information	27-9

Chapter 28

Situations to Avoid

Chapter Overview	28-3
Situations to Avoid in Administering OnLine	28-3

Monitoring and Performance

Chapter 29

Monitoring OnLine

Chapter Overview	29-5
Sources of Information for Monitoring OnLine	29-6
What Is the Message Log?	29-6
What Is the Console?	29-7
Monitoring Using ON-Monitor	29-8
Monitoring Using SMI Tables	29-8
Monitoring Using onstat and oncheck Utilities	29-8
Monitoring Configuration Information	29-9
Monitoring Checkpoint Information	29-11
Monitoring Shared Memory	29-12
Monitoring Shared-Memory Segments	29-13
Monitoring Shared-Memory Profile	29-13
Monitoring Buffers	29-15
Monitoring Buffer-Pool Activity	29-19
Monitoring Latches	29-22

Monitoring Locks	29-23
Monitoring Active Tbspaces	29-26
Monitoring Virtual Processors	29-27
Monitoring Sessions and Threads	29-29
Monitoring Transactions	29-33
Monitoring Databases	29-36
Monitoring Logging Activity	29-37
Monitoring Logical-Log Files	29-37
Monitoring the Physical-Log File	29-40
Monitoring the Physical- and Logical-Log Buffers	29-41
Monitoring Chunk Status	29-43
Monitoring Disk Usage	29-46
Monitor Chunks	29-46
Monitoring Tbspaces and Extents	29-50
Monitoring Blobs in a BlobSpace	29-53
Monitoring Blobs in a DbSpace	29-56
Monitoring Data-Replication Status	29-58

Chapter 30

Improving Performance

Chapter Overview	30-3
Areas of OnLine Operation That Affect Performance	30-3
Guidelines Depend on Your Application	30-3
When Is Tuning Needed?	30-4
Configuring Disk Space	30-6
Recommendations	30-6
Specifying Where Sorting Occurs	30-7
Psort (Parallel-Sort) Package	30-8
Configuring and Using Shared Memory	30-10
Avoiding Resource Bottlenecks	30-10
Allocating Shared-Memory Buffers	30-11
Log Buffer Size	30-12
Page-Cleaner Parameters	30-13
Checkpoint Frequency	30-16
Shared-Memory Resources	30-17
Configuring Virtual Processors	30-18
Configuring CPU Virtual Processors	30-18
Configuring AIO Virtual Processors	30-19
Configuring Network Virtual Processors	30-20
Setting USEOSTIME	30-20

Data Migration

Chapter 31

Data Migration

- Chapter Overview 31-3
- Moving Databases and Tables 31-4
 - Summary of Methods for Moving Data 31-4
- Using the **onunload** and **onload** Utilities 31-5
 - Steps for Using **onunload** and **onload** 31-7
- Choosing Between **dbload**, **dbimport**, and **LOAD** 31-8
- Creating a New Database Object 31-9
- Modifying the Database Schema 31-10
- Using the **UNLOAD** and **LOAD** Statements 31-10
 - Steps for Using **UNLOAD** and **LOAD** 31-11
- Using the **dbload** Utility 31-11
 - Steps for Using **dbload** 31-12
- Using the **dbexport** and **dbimport** Utilities 31-13
 - Steps for Using **dbexport/dbimport** 31-13
- Migrating Data from OnLine to INFORMIX-SE 31-14
 - Remove OnLine Specifics from the Schema File 31-14
 - Alert Users to OnLine and SE Differences 31-14
- Moving Data from SE to OnLine 31-16
 - Add OnLine Specifics to the Schema File 31-16
 - Alert Users to OnLine and SE Differences 31-16
- Moving Data from One Locale to Another 31-17
 - Character Types with NLS Databases 31-17
 - The Locale of an NLS Database 31-17
 - Steps for Moving Data to an NLS Database 31-17
 - Steps for Moving Data from an NLS Database 31-18

Distributed Data

Chapter 32

What Is Two-Phase Commit?

- Chapter Overview 32-3
- Two-Phase Commit Protocol 32-3
 - When Is the Two-Phase Commit Protocol Used? 32-3
 - What Goals Does the Two-Phase Commit Protocol Achieve? 32-5
 - Two-Phase Commit Concepts 32-5
 - Phases of the Two-Phase Commit Protocol 32-6
 - Examples of Two-Phase Commit Transactions 32-7
 - How the Two-Phase Commit Protocol Handles Failures 32-9
 - Presumed-Abort Optimization 32-17
- Independent Actions 32-18

What Initiates Independent Action	32-18
Possible Results of Independent Action	32-19
The Heuristic Rollback Scenario	32-21
The Heuristic End-Transaction Scenario	32-25
Tracking a Global Transaction	32-27
Two-Phase Commit Protocol Errors	32-27
Two-Phase Commit and Logical Log Records	32-28
Logical-Log Records When the Transaction Commits	32-28
Logical-Log Records Written During a Heuristic Rollback	32-30
Logical-Log Records Written After a Heuristic End Transaction	32-32
Configuration Parameters Used in Two-Phase Commits	32-34
Function of the DEADLOCK_TIMEOUT Parameter	32-34
Function of the TXTIMEOUT Parameter	32-34

Chapter 33 **How to Manually Recover from Failed Two-Phase Commit Protocol**

Chapter Overview	33-3
Procedure to Determine If Manual Recovery Is Required	33-3
Determine Whether a Transaction Was Implemented Inconsistently	33-4
Determine If the Networked Database Contains Inconsistent Data	33-5
Decide If Action Is Needed to Correct the Situation	33-8
Example of Manual Recovery	33-9

Reference

Chapter 34 **ON-Monitor**

Chapter Overview	34-3
Using ON-Monitor	34-3
Help and Navigation Within ON-Monitor	34-4
Executing Shell Commands from Within ON-Monitor	34-4
ON-Monitor Screen Options	34-4

Chapter 35 **OnLine Configuration Parameters**

Chapter Overview	35-5
ONCONFIG Parameters	35-5
ONCONFIG File Conventions	35-6
ADTERR	35-6
ADTMODE	35-7
ADTPATH	35-7
ADTSIZE	35-7
AFF_NPROCS	35-8
AFF_SPROC	35-8
BUFFERS	35-8
CHUNKS	35-9

CKPTINTVL	35-10
CLEANERS	35-11
CONSOLE	35-11
DBSERVERALIASES	35-11
DBSERVERNAME	35-12
DBSPACES	35-13
DBSPACETEMP	35-13
DEADLOCK_TIMEOUT	35-14
DRAUTO	35-14
DRINTERVAL	35-15
DRLOSTFOUND	35-15
DRTIMEOUT	35-16
DUMPCNT	35-16
DUMPCORE	35-17
DUMPDIR	35-17
DUMPGCORE	35-18
DUMPSHMEM	35-18
FILLFACTOR	35-19
LOCKS	35-19
LOGBUFF	35-20
LOGFILES	35-20
LOGSIZE	35-21
LOGSMAX	35-21
LRUS	35-22
LRU_MAX_DIRTY	35-22
LRU_MIN_DIRTY	35-23
LTAPEBLK	35-23
LTAPEDEV	35-24
LTAPESIZE	35-24
LTXEHWM	35-25
LTXHWM	35-25
MIRROR	35-26
MIRROROFFSET	35-26
MIRRORPATH	35-26
MSGPATH	35-27
MULTIPROCESSOR	35-27
NETTYPE	35-28
NOAGE	35-30
NUMAIOVPS	35-30
NUMCPUVPS	35-30
OFF_RECVRY_THREADS	35-31
ON_RECVRY_THREADS	35-31
PHYSBUFF	35-32

PHYSDBS	35-33
PHYSFILE	35-33
RA_PAGES	35-34
RA_THRESHOLD	35-34
RESIDENT	35-35
ROOTNAME	35-35
ROOTOFFSET	35-35
ROOTPATH	35-36
ROOTSIZE	35-36
SERVERNUM	35-37
SHMADD	35-37
SHMBASE	35-38
SHMTOTAL	35-38
SHMVIRTSIZE	35-39
SINGLE_CPU_VP	35-40
STACKSIZE	35-40
STAGEBLOB	35-41
TAPEBLK	35-41
TAPEDEV	35-42
TAPESIZE	35-44
TBLSPACES	35-44
TRANSACTIONS	35-45
TXTIMEOUT	35-45
USEOSTIME	35-46
USERTHREADS	35-46

Chapter 36

The sysmaster Database

Chapter Overview	36-3
What Is the sysmaster Database?	36-3
Using the System-Monitoring Interface	36-4
What are the SMI Tables?	36-4
Accessing SMI Tables	36-5
The System-Monitoring Interface Tables	36-7
sysadinfo	36-8
sysaudit	36-9
syschkio	36-9
syschunks	36-10
sysdatabases	36-11
sysdbspaces	36-12
sysdri	36-13
sysextents	36-13
syslocks	36-13
syslogs	36-14

sysprofile	36-15
sysptprof	36-16
sysseprof	36-17
sysessions	36-18
syseswts	36-20
systabnames	36-20
sysvpprof	36-21
The SMI Tables Map	36-21
Using SMI Tables to Obtain onstat Information	36-24

Chapter 37

OnLine Utilities

Chapter Overview	37-5
oncheck : Check, Repair, or Display	37-6
Syntax	37-8
Option Descriptions	37-9
oninit : Initialize OnLine	37-16
Syntax	37-16
Initialize Shared Memory Only	37-16
Initialize Disk Space and Shared Memory	37-17
onload : Create a Database or Table	37-18
Syntax	37-18
Specify Source Parameters	37-19
Create Options	37-20
Constraints That Affect onload and onunload	37-20
Logging While Using onload	37-21
BlobSpace Blobs Relocation	37-22
onlog : Display Logical-Log Contents	37-23
Syntax	37-23
Log-Record Read Filters	37-24
Log-Record Display Filters	37-25
onmode : Mode and Shared-Memory Changes	37-27
Syntax	37-28
Change OnLine Modes	37-29
Force a Checkpoint	37-30
Change Shared-Memory Residency	37-30
Switch the Logical-Log File	37-31
Kill an OnLine Session	37-31
Kill an OnLine Transaction	37-32
Set Data-Replication Types	37-32
Add a Shared-Memory Segment	37-34
Add or Remove Virtual Processors	37-34
Change Database Format	37-35
Regenerate .infos File	37-36

onparams: Modify Log-Configuration Parameters	37-37
Syntax	37-37
Add a Logical-Log File	37-38
Drop a Logical-Log File	37-38
Change Physical-Log Parameters	37-39
onspaces: Modify Blobspaces or Dbspaces	37-40
Syntax	37-40
Create a Blobspace or Dbspace	37-41
Drop a Blobspace or Dbspace	37-42
Add a Chunk	37-42
Drop a Chunk	37-43
Start Mirroring	37-44
End Mirroring	37-45
Change Chunk Status	37-45
onstat: Monitor OnLine Operation	37-46
Syntax	37-47
Output Header	37-48
Option Descriptions	37-49
ontape: Logging, Archives, and Restore	37-70
Syntax	37-71
Archive Data Managed by an OnLine Database Server	37-72
Change Database Logging Status	37-73
Back up Logical-Log Files	37-74
Start Continuous Backup of Logical-Log Files	37-74
Restore Data from an Archive	37-75
Preparing for Data Replication	37-76
onunload: Transfer Binary Data in Page Units	37-77
Syntax	37-77
Constraints That Affect onunload	37-78
Unloading a Database or Table	37-78
Logging Mode	37-79
Locking During Unload Operation	37-79

Chapter 38

OnLine Message Log Messages

Chapter Overview	38-3
How the Messages Are Ordered in This Chapter	38-3
Message Categories	38-4
Messages: A-B	38-4
Messages: C	38-6
Messages: D-E-F	38-11
Messages: G-H-I	38-13
Messages: J-K-L-M	38-15
Messages: N-O-P	38-17

Messages: Q-R-S 38-20
Messages: T-U-V 38-23
Messages: W-X-Y-Z 38-26
Messages: Symbols 38-27

Chapter 39

Interpreting Logical-Log Records

Chapter Overview 39-3
Reading Logical-Log Records 39-3
 Transactions That Drop a Table or Index 39-4
 Transactions That Are Rolled Back 39-4
 Checkpoints with Active Transactions 39-4
 Distributed Transactions 39-5
Logical-Log Record Structure 39-5
 Logical-Log Record Header 39-6
 Logical-Log Record Types and Additional Columns 39-7

Chapter 40

OnLine Disk Structure and Storage

Chapter Overview 40-3
Dbspace Structure and Storage 40-4
 Structure of the Root Dbspace 40-4
 Structure of a Regular Dbspace 40-13
 Structure of a Mirror Chunk 40-15
 Structure of the Chunk Free-List Page 40-16
 Structure of the Tblspace Tblspace 40-17
 Structure of the Database Tblspace 40-20
 Structure of a Dbspace Bit-Map Page 40-22
 Structure and Allocation of an Extent 40-24
 Structure and Storage of a Dbspace Page 40-30
 Structure of Index Pages 40-43
Blobspace Structure and Storage 40-54
 Structure of a Blobspace 40-54
 Blob Storage and the Blob Descriptor 40-56
 Structure of a Dbspace Blob Page 40-57
 Blobspace Page Types 40-59
 Structure of a Blobspace Blobpage 40-60
Database and Table Creation: What Happens on Disk 40-63
 Creating a Database 40-63
 Creating a Table 40-64

Chapter 41

OnLine Environment Variables

Chapter Overview 41-3
Environment Variable Used During Initialization 41-3
Environment Variables Sent by the Client 41-3

Chapter 42

Files Used by OnLine

Chapter Overview	42-3
Descriptions of Files	42-4
af.xxx	42-4
ARCreqid.NOT	42-4
buildsmi.xxx	42-5
config.arc	42-5
core	42-5
gcore.xxx	42-5
informix.rc	42-5
~/informix	42-5
.inf.servicename	42-6
.infos.dbservername	42-6
The Message Log	42-6
oncatlgr.out.pidnum	42-6
onconfig.std	42-7
onconfig	42-7
The ONCONFIG File	42-7
oncfg_servername.servernum	42-8
oper_deflt.arc	42-8
shmem.xxx	42-8
sqlhosts	42-8
status_vset_volnum.itgr	42-9
sysfail.pidnum	42-9
tctermcap	42-9
VP.servername.xxC	42-9
A Sample onconfig.std File	42-10

Index

Introduction

INFORMIX-OnLine Dynamic Server and Other Informix Products	3
Other Useful Documentation	4
How to Use This Manual	5
Typographical Conventions	5
Command-Line Conventions	5
Example Code Conventions	8
Useful On-Line Files	8
ASCII and PostScript Error Message Files	9
The Demonstration Database	9
Creating the Demonstration Database on INFORMIX-OnLine Dynamic Server	10
New Features in INFORMIX-OnLine Dynamic Server Version 6.0	11

INFORMIX-OnLine Dynamic Server is a database server that combines high-availability, on-line transaction-processing (OLTP) performance with multimedia capabilities. By managing its own shared-memory resources and disk I/O, **INFORMIX-OnLine Dynamic Server** delivers process concurrency while maintaining transaction isolation. Table data can span multiple disks, freeing administrators from constraints imposed by data-storage limitations.

The functionality that provides for client/server communications and for distributed database access is an integral part of **INFORMIX-OnLine Dynamic Server**.

The additional support provided by **INFORMIX-OnLine/Optical** enables data storage on an optical subsystem. The **INFORMIX-TP/XA** product allows you to use the **OnLine** database server as a Resource Manager within an X/Open environment.

INFORMIX-OnLine Dynamic Server and Other Informix Products

Informix Software produces a variety of application development tools, CASE tools, database servers, and utilities. **DB-Access** is a utility that allows you to access, modify, and retrieve information from **OnLine** relational databases. **INFORMIX-OnLine Dynamic Server** supports all application development tools currently available, including products like **INFORMIX-SQL**, **INFORMIX-4GL** and the **Interactive Debugger**, and the Informix SQL API products, such as **INFORMIX-ESQL/C**. If you are using an optical-storage subsystem for multimedia data, you access the data with the **INFORMIX-OnLine/Optical** product.

Other Useful Documentation

You might want to refer to a number of related Informix product documents that complement the *INFORMIX-OnLine Dynamic Server Administrator's Guide*.

- The *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* describes the archiving process and the tools and commands available for making archives of your **INFORMIX-OnLine Dynamic Server** databases and backups of logical logs.
- The *INFORMIX-OnLine Dynamic Server Trusted Facility Manual* describes the secure auditing capabilities of **INFORMIX-OnLine Dynamic Server**, including the creation and maintenance of audit logs.
- You might find it convenient to use the *INFORMIX-OnLine Dynamic Server Quick Reference Guide* for a summary of the ON-Monitor menu options and their command-line equivalents.
- If you have never used Structured Query Language (SQL) or an Informix application development tool, read the *Informix Guide to SQL: Tutorial*. The manual describes the fundamental ideas and terminology that are used when planning, using, and implementing a relational database.
- A companion volume to the Tutorial, the *Informix Guide to SQL: Reference*, provides reference information on the types of databases you can create, the data types supported by Informix products, the system catalog tables associated with a database, environment variables, and the SQL utilities. This guide also provides a detailed description of the **stores6** demonstration database and contains a glossary.
- An additional companion volume to the Reference, the *Informix Guide to SQL: Syntax*, provides a detailed description of all the SQL statements supported by Informix products. This guide also provides a detailed description of Stored Procedure Language (SPL) statements.
- You, or whoever installs **OnLine**, should refer to the *UNIX Products Installation Guide* for your particular release to ensure that **OnLine** is properly set up before you begin to work with it. A matrix depicting possible client/server configurations is included in the *Installation Guide*.
- The *DB-Access User Manual* describes how to invoke the utility to access, modify, and retrieve information from **OnLine** relational databases.
- When errors occur, you can look them up, by number, and find their cause and solution in the *Informix Error Messages* manual. If you prefer, you can



look up the error messages in the on-line message file described in the section “ASCII and PostScript Error Message Files” later in this Introduction.

How to Use This Manual

This section describes the typographical, command-line, and example code conventions used in the *INFORMIX-OnLine Dynamic Server Administrator's Guide* and other Informix product documentation. For readability within this manual, **INFORMIX-OnLine Dynamic Server** is often referred to as **OnLine**.

Typographical Conventions

The *INFORMIX-OnLine Dynamic Server Administrator's Guide* uses a standard set of conventions to introduce new terms, illustrate screen displays, describe command syntax, and so forth. The following typographical conventions are used throughout the manual:

<i>italics</i>	New terms, emphasized words, and variables are printed in italics.
boldface	Database names, table names, column names, filenames, utilities, and other similar terms are printed in boldface.
computer	Information that OnLine displays and information that you enter are printed in a computer typeface.
KEYWORD	All keywords appear in uppercase letters.
	This symbol indicates a unique identifier (primary key) for each table.
	This symbol indicates a warning. Warnings provide critical information that, if ignored, could cause harm to your database.

Additionally, when you are instructed to “enter” or “execute” text, immediately press RETURN after the entry. When you are instructed to “type” the text, no RETURN is required.

Command-Line Conventions

OnLine supports a variety of command-line options. These are commands that you enter at the operating-system prompt to perform certain functions as part of **OnLine** administration.

This section defines and illustrates the format of the commands. These commands have their own conventions, which may include alternative forms of a command, required and optional parts of the command, and so forth.

Each diagram displays the sequences of required and optional elements that are valid in a command. A diagram begins at the upper left with a command. It ends at the upper right 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.

Along a command-line path, you might encounter the following elements:

command This required element is usually the product name or other short word used to invoke the product or call 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 must use lowercase letters.

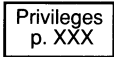
variable A word in italics represents a value that you must supply, such as a database, file, or program name. The nature of the value is explained immediately following the diagram.

-flag A flag is usually an abbreviation for a function, menu, or option name or for a compiler or preprocessor argument. You must enter a flag exactly as shown, including the preceding hyphen.

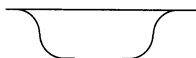
.ext A filename extension, such as **.sql** or **.cob**, might follow a variable representing a filename. Type this extension exactly as shown, immediately after the name of the file and a period. The extension might be optional in certain products.

(.,;+*-/) Punctuation and mathematical notations are literal symbols that you must enter exactly as shown.

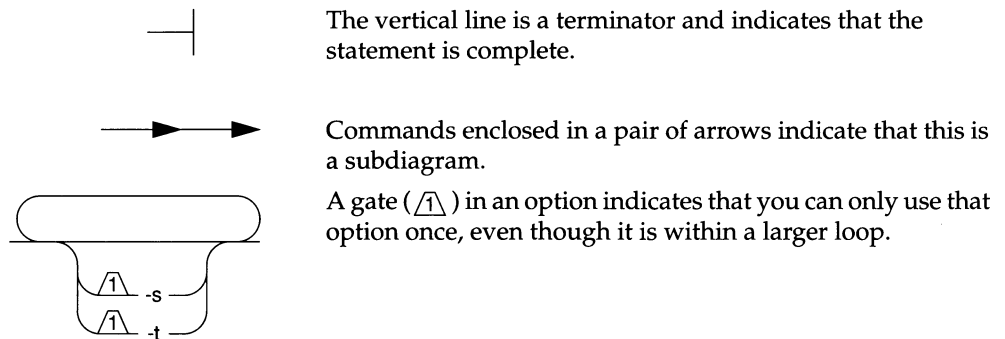
' ' and " " Single and double quotes are literal symbols that you must enter as shown.

 A reference in a box represents a subdiagram on the same page or another page. Imagine that the subdiagram is spliced into the main diagram at this point.

— ALL — A shaded option is the default. Even if you do not explicitly type the option, it will be in effect unless you choose another option.



A branch below the main line indicates an optional path.



The following diagram shows the flow of the **onunload** utility command. To learn more about using the **onunload** utility, see “onunload: Transfer Binary Data in Page Units” on page 37-77.

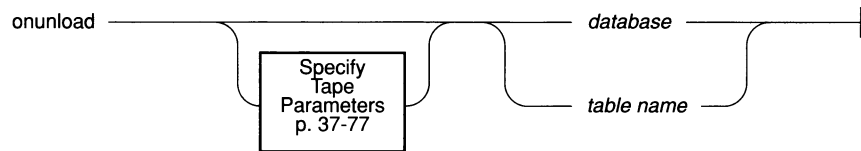


Figure 1 *Elements of a command-line diagram*

To construct a correct command, start at the top left with the command `onunload`. Then follow the diagram to the right, including the elements that you want. This diagram conveys the following information:

1. You must type the word **onunload**.
2. You can change the parameters of the tape device that is to receive the data. If you wish to do this, turn to “Specify Destination Parameters” on page 37-77 for further syntax information.
3. You must specify either a database name or a table name.
4. After you choose the database name or table name, you come to the terminator. Your **onunload** command is complete. Press RETURN to execute the command.

Example Code Conventions

Some examples of SQL code occur in this manual. Except where noted, the code is not specific to any single Informix application development tool. If only SQL statements are listed in the example, they are not delineated by semicolons. To use this SQL code for a specific product, you must apply the syntax rules for that product. For example, if you are using the Query Language option of **DB-Access** or **INFORMIX-SQL**, you must delineate the statements with semicolons. If you are using an embedded language, you must use EXEC SQL and a semicolon (or other appropriate delimiters) at the start and end of each statement, respectively.

For example, you might see the following example code:

```
CONNECT TO stores6
.
.
.
DELETE FROM customer
      WHERE customer_num = 121
.
.
.
COMMIT WORK
DISCONNECT CURRENT
```

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

Also note that dots in the example indicate that more code would be added in a full application, but it is not necessary to show it to describe the concept being discussed.

Useful On-Line Files

In addition to the Informix set of manuals, the following on-line files, located in the **\$INFORMIXDIR/release** directory, might supplement the information in the *INFORMIX-OnLine Dynamic Server Administrator's Guide*:

Documentation Notes describe features not covered in the manual or that have been modified since publication. The file containing the Documentation Notes for this product is called **ONLINEDOC_6.0**.

- Release Notes** describe feature differences from earlier versions of Informix products and how these differences might affect current products. The file containing the Release Notes for this product is called **SERVERS_6.0**.
- Machine Notes** describe any special actions required to configure and use Informix products on your machine. The file containing the Machine Notes for this product is called **ONLINE_6.0**.

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

ASCII and PostScript Error Message Files

Informix software products provide ASCII files that contain all the Informix error messages and their corrective actions. To access 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**). See the Introduction to the *Informix Error Messages* manual for a detailed description of these scripts.

The optional **Informix Messages and Corrections** product provides PostScript files that contain the error messages and their corrective actions. If you have installed this product, you can print the PostScript files on a PostScript printer. The PostScript error messages are distributed in a number of files of the format **errmsg1.ps**, **errmsg2.ps**, and so on. These files are located in the **\$INFORMIXDIR/msg** directory.

The Demonstration Database

The **DB-Access** utility, which is provided with your Informix database server products, includes a demonstration database called **stores6** that contains information about a fictitious wholesale sporting-goods distributor. The sample command files that make up a demonstration application are also included.

Most of the examples in this manual are based on the **stores6** demonstration database. The **stores6** 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 **dbaccessdemo6** and is located in the **\$INFORMIXDIR/bin** directory. The database name that you supply is the name given to the demonstration database. If you do not supply a database name, the name defaults to **stores6**. Follow these rules for naming your database:

- Names for databases can be up to 18 characters long for **INFORMIX-OnLine Dynamic Server** databases.
- The first character of a name must be a letter or an underscore (_).
- You can use letters, characters, and underscores (_) for the rest of the name.
- **DB-Access** makes no distinction between uppercase and lowercase letters.
- The database name should be unique.

When you run **dbaccessdemo6**, you are, as the creator of the database, the owner and Database Administrator (DBA) of that database.

If you installed your Informix database server product according to the installation instructions, the files that make up the demonstration database are protected so you cannot make any changes to the original database.

You can run the **dbaccessdemo6** script again whenever you want to work with a fresh demonstration database. The script prompts you when the creation of the database is complete, and asks if you would like to copy the sample command files to the current directory. Enter "N" if you have made changes to the sample files and do not want them replaced with the original versions. Enter "Y" if you want to copy over the sample command files.

Creating the Demonstration Database on INFORMIX-OnLine Dynamic Server

Use the following steps to create and populate the demonstration database:

1. Set the **INFORMIXDIR** environment variable so that it contains the name of the directory in which your Informix products are installed. Set **INFORMIXSERVER** to the name of the default database server. The name of the default database server must exist in the **\$INFORMIXDIR/etc/sqlhosts** file. (For a full description of environment variables, see Chapter 4 of the

Informix Guide to SQL: Reference. For more information about the `sqlhosts` file, see Chapter 4, "Configuring Connectivity."

2. Create a new directory for the SQL command files. Create the directory by entering the following command:

```
mkdir dirname
```

3. Make the new directory the current directory by entering the following command:

```
cd dirname
```

4. Create the demonstration database and copy over the sample command files entering the command `dbaccessdemo6`.

To create the demonstration database without logging enter the following command:

```
dbaccessdemo6 dbname
```

To create the demonstration database with logging enter the following command:

```
dbaccessdemo6 -log dbname
```

The data for the database is put into the root dbspace.

To give someone else the permissions to access the command files in your directory, use the UNIX `chmod` command.

To give someone else access to the database that you have created, grant them the appropriate privileges using the GRANT statement in **DB-Access**. To remove privileges, use the REVOKE statement. The GRANT and REVOKE statements are described in Chapter 1 of the *Informix Guide to SQL: Syntax*.

New Features in INFORMIX-OnLine Dynamic Server Version 6.0

The Introduction to each Version 6.0 product manual contains a list of new features for that product. The Introduction to each manual in the Version 6.0 *Informix Guide to SQL* series contains a list of new SQL features.

A comprehensive listing of all the new features for Version 6.0 Informix products is found in the Release Notes file called `SERVERS_6.0`.

This section highlights the major new features implemented in Version 6.0 of **INFORMIX-OnLine Dynamic Server**.

- Client/Server Communications

The functionality that provides remote client/server communications is now an integral part of all Version 6.0 Informix products. Any Version 6.0 product can make network connections as well as local connections with any other Version 6.0 product. In addition, the Version 6.0 communication facilities allow network connections to Informix Version 4.1 and Version 5.x products. The Version 6.0 product release does not include **INFORMIX-NET**, **INFORMIX-STAR**, or **INFORMIX-NET/Relay Module** because their functionality is now provided with each Informix product.

- Parallel, on-line Archive, Backup, Restore, and Tape-Management Facilities

A new utility, called ON-Archive, provides functionality similar to that provided by **ontape** for archiving and restoring **OnLine** databases and for backing up logical-log files.

In addition to providing functionality similar to **ontape**, ON-Archive can do the following:

- Archive portions of a database (one or more dbspaces).
- Use multiple tape drives simultaneously (parallel archives and restores).
- Provide sophisticated tape-management facilities.
- Provide capabilities for unattended operation.
- Allow you to track and schedule archives and backups.
- Provide facilities for data compression and encryption.

Both ON-Archive and the **ontape** utility now allow administrators to restore single or multiple dbspaces and both utilities are able to process multiple logical log records in parallel, so recovery occurs more quickly.

- On-line operation with down dbspaces

When an error is detected in one dspace managed by **OnLine** and that dspace does not contain the root, physical, or logical logs, only that dspace is inaccessible. **OnLine** remains in on-line mode, and other dbspaces that are not affected remain available for use.

- System-Monitoring Interface

The system-monitoring interface (SMI) provides tables of information based on structures in shared memory that users and **OnLine** administrators can query using SQL. SMI contains information on users and user

actions such as writes and deletes, locks and wait times, extents, chunks, dbspaces, tables, databases, and the use of logical-log files.

- **Data Replication**

OnLine offers a nearly transparent way of replicating data across a network, allowing organizations to maintain a backup copy of an entire **OnLine** database server at another site. This feature allows administrators to automatically or manually direct users to the secondary database server after a failure of the primary database server, dramatically reducing the amount of time spent in recovery. The secondary database server also allows read-only access, providing the opportunity for load balancing (that is, OLTP applications on the primary database server, reports and queries on the secondary database server).

- **Dynamic, Scalable Architecture**

The dynamic scalable architecture provides a flexible threading architecture for both on-line transaction processing (OLTP) and decision-support environments. For OLTP environments, a small number of database server processes efficiently service a much larger number of user sessions. For decision support and batch jobs, a single user session can efficiently spawn multiple threads that run in parallel, thereby using computer resources more effectively.

- **Enhanced Mirroring**

Mirroring pairs a chunk of one defined dbspace or blobspace with a mirror chunk. Every write to the primary chunk is accompanied by an identical write to the mirror chunk. If a failure occurs on the primary chunk, mirroring enables users to read from and write to the mirror chunk and therefore, stay on-line while you recover from the primary chunk.

- **C2-level secure auditing.**

C2-level auditing creates a record of selected user activities. Auditing can detect users attempting unauthorized accesses of the database, assess potential security damage if unusual activity occurs, provide evidence, if necessary, and provide a deterrent against unwanted activities.

The **OnLine** auditing facility is designed to meet the C2 class of trust as specified in the *Trusted Computer System Evaluation Criteria* (CSC-STD-001-83) and the *Trusted Database Interpretation* (NCSC-TG-021), both published by the U.S. Department of Defense.

- **Nonroot Temporary Dbspace**

OnLine administrators and users can now specify that temporary tables be built in a temporary dbspace. These temporary dbspaces are ignored during a full-system archive.

- **Removal of Limits**

The maximum number of users, logical-log files, chunks, dbspaces, databases, and buffers in **OnLine** are substantially higher.
- **OnLine Utility Renaming**

The **OnLine** utilities (**tbstat**, **tbinit**, **tbmode**, **tblog**, **tbcheck**, **tbparams**, **tbspaces**, **tbmonitor**, **tbload**, and **tbunload**) have been renamed to replace “tb” with “on” (**onstat**, **oninit**, **onmode**, **onlog**, **oncheck**, **onparams**, **onspaces**, **onmonitor**, **onload**, and **onunload**). DB-Monitor is now called ON-Monitor.
- **OnLine Utility Enhancements**

The **OnLine** utilities have been enhanced so you can perform the following actions:

 - Drop an empty chunk
 - Start or end mirroring for an existing space from the command line
 - Change the logging status of a database to unbuffered logging and to ANSI-compliant
 - Interactively display statistics derived from shared memory
 - Display information about users or transactions
 - Resize logical-log files



What Is INFORMIX-OnLine Dynamic Server?





What Is INFORMIX- OnLine Dynamic Server?

Chapter Overview 3

What Is OnLine? 3

Client/Server Architecture 4

The Client/Server Connection 4

High Performance 5

Dynamic Tuning 5

Raw Disk Management 5

Dynamic Shared-Memory Management 6

Dynamic Thread Allocation 6

Fault Tolerance and High Availability 6

Archives and Backups of Transaction Records 6

Fast Recovery 7

Mirroring 7

Data Replication 7

Multimedia Support 8

Distributed Data Queries 8

Database Server Security 8

Who Uses OnLine? 9

End Users 9

Application Developers 9

Database Administrators 10

OnLine Administrators 10

OnLine Operators 10

Features Beyond the Scope of OnLine	10
No Bad-Sector Mapping	10
No Blob Scanning or Compression	11

Chapter Overview

This chapter introduces the **INFORMIX-OnLine Dynamic Server**, and includes the following sections:

- What is **OnLine**?
- Who uses **OnLine**?
- Features beyond the scope of **OnLine**

These sections briefly describe **OnLine** and point out where you can find more detailed information elsewhere in this document, or in other documents.

What Is OnLine?

OnLine is a *database server*. A database server is a software package that manages access to one or more databases for one or more client applications. It is the principal component of a database management system. Specifically, **OnLine** is a database server in a *relational* database management system (RDBMS). A relational database is one in which the data is organized in tables that consist of rows and columns.

The **OnLine** database server offers the following features:

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

Each of these features is explained in the following sections.

Client/Server Architecture

OnLine is a *server* for client applications. More specifically, **OnLine** 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 **OnLine**. Client programs include the **DB-Access** utility, embedded language programs such as an **INFORMIX-ESQL/C** program, and **INFORMIX-4GL** programs.

Client processes are independent of **OnLine** processes. Database users run client applications as they need to access information. **OnLine** processes are started by the **OnLine** administrator and they are presumed to execute continuously during the period that users access the databases. See Chapter 12, “What Is the Dynamic Scalable Architecture?,” for a description of the **OnLine** processes and the methods by which they serve client applications.

The Client/Server Connection

A client application communicates with **OnLine** through the connection facilities that **OnLine** provides. These facilities are fully described in Chapter 4, “Configuring Connectivity.”

At the source-code level, a client connects to **OnLine** through an SQL statement. Beyond that, the client’s use of **OnLine** 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 **OnLine**.

You, as the **OnLine** administrator, specify the types of connections that **OnLine** supports in a connectivity information file, called **sqlhosts**. The **sqlhosts** file contains the names of each of the database servers (called the *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 **OnLine** through an SQL statement, the client transparently accesses this information and makes the connection using the specified protocol.

High Performance

OnLine achieves high performance through the following mechanisms:

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

Each of these mechanisms is explained in the following paragraphs.

Dynamic Tuning

OnLine's *Dynamic Scalable Architecture* (DSA) enables you to add both processes and shared-memory while **OnLine** is in on-line mode. **OnLine's** dynamic scalable architecture is described in Chapter 12, "What Is the Dynamic Scalable Architecture?" **OnLine's** use of shared memory is described in Chapter 14, "OnLine Shared Memory." Chapter 30, "Improving Performance," offers recommendations for tuning **OnLine's** performance.

Raw Disk Management

OnLine can use both UNIX file system disk space and raw disk space. When using raw disk space, however, **OnLine** 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, **OnLine** can manage the physical organization of data and minimize disk I/O. Doing so results in three performance advantages:

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

If these things are not a primary concern, you can also configure **OnLine** to use regular operating system files to store data. In this case, **OnLine** manages the file contents but the operating system manages the I/O. See Chapter 10, "Where Is Data Stored?," for more information about **OnLine's** use of disk space.

Dynamic Shared-Memory Management

All applications that use a single instance of an **OnLine** database server share data in the server's memory space. After one application reads data from a table, other applications can access whatever data is already in memory. Disk access might not be necessary.

OnLine 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. **OnLine** adds memory dynamically as it needs it and you, as the administrator, can also add segments to shared memory, if necessary. See Chapter 15, "Managing OnLine Shared Memory," for information on how to add a segment to **OnLine** shared memory.

Dynamic Thread Allocation

OnLine supports multiple client applications using a relatively few number of processes called virtual processors. A virtual processor is a multithreaded process that serves multiple clients and, in some cases, runs multiple threads to work in parallel for a single client. In this way, **OnLine** provides a flexible architecture that is well-suited for both on-line transaction processing (OLTP) and for decision-support applications. See Chapter 12, "What Is the Dynamic Scalable Architecture?," for a description of **OnLine** Dynamic Scalable Architecture.

Fault Tolerance and High Availability

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

- Archives and backups of transaction records
- Fast recovery
- Mirroring
- Data replication

Archives and Backups of Transaction Records

OnLine provides you with the ability to *archive* the data it manages and also store (*back up*) changes to the database server and data since the archive was performed. The changes are stored in *logical-log files*.

As explained in the book devoted to the topic, *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*, **OnLine** allows you to create the archive tapes and the logical-log backup tapes while users are accessing **OnLine**. You can also use on-line archiving to create incremental archives. Incremental archiving enables you to only back up data that has changed since the last archive, which reduces the amount of time required for archiving.

After a media failure, if critical data was not damaged (and **OnLine** 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 **OnLine** starts up, it checks to see if the physical log is empty because that implies that **OnLine** shut down in a controlled fashion. If the physical log is *not* empty, **OnLine** automatically performs an operation called *fast recovery*. Fast recovery automatically restores **OnLine** databases to a state of physical and logical consistency after a system failure that might leave one or more transactions uncommitted. During fast recovery, **OnLine** 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

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

Mirroring

When you use the **OnLine** enhancement that performs disk mirroring, **OnLine** writes each piece of data in two places. When data is mirrored in this way, it can eliminate data loss as a result of media (hardware) crashes. If mirrored data becomes unavailable for any reason, the mirror of the data is accessed immediately and transparently to users. Mirroring is explained in Chapter 23, "What Is Mirroring?"

Data Replication

If your organization requires a very high degree of availability, you can replicate **OnLine** and its databases, running simultaneously, on a second computer. Replicating **OnLine** and its databases protects the two database servers

from catastrophic failure; if one site experiences a disaster, applications can be directed immediately to use the second database server in the pair. See Chapter 25, “What Is Data Replication?”

Multimedia Support

OnLine supports two *blob* (binary large object) data types – TEXT and BYTE – which place no practical limit on the size of the stored data item. **OnLine** stores this blob data either with other database data or in specially designated portions of the disk called blobspaces.

For more information about storing blob data on write-once-read-many (WORM) optical devices, refer to the *INFORMIX-OnLine/Optical User Manual*.

Distributed Data Queries

OnLine allows users to query and update more than one database across multiple **OnLine** database servers within a single transaction. The **OnLine** database servers can reside within a single host computer or on the same network. **OnLine** supports both TCP/IP and IPX/SPX networks. A two-phase commit protocol ensures that transactions are uniformly committed or rolled back across the multiple database servers. The protocol is described in detail in Chapter 32, “What Is Two-Phase Commit?”

You can use **INFORMIX-Gateway with DRDA** to make distributed queries that involve both Informix and DRDA-compliant databases. For more information, refer to the *INFORMIX-Gateway with DRDA User Manual*.

You can also use **OnLine** in a heterogeneous environment that conforms to X/Open. For more information about using **OnLine** within an X/Open environment, refer to the *INFORMIX-TP/XA User Manual*.

Database Server Security

The databases and tables managed by **OnLine** enforce access based on a set of database and table privileges, which are managed through the use of GRANT and REVOKE SQL statements. They are explained in the *Informix Guide to SQL: Tutorial* and *Informix Guide to SQL: Syntax*.

In addition to this type of security, **OnLine** offers the ability to audit database events on a database-server wide basis. Auditing, described in the *INFORMIX-OnLine Dynamic Server Trusted Facility Manual*, enables you to

track which users performed which actions to which objects at what time. This information can be used to monitor database activity for suspicious use, deter unscrupulous users, or even act as evidence of database server abuse.

Who Uses OnLine?

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

- End users
- Application developers
- Database administrators
- **OnLine** administrators
- **OnLine** operators

End Users

End users access, insert, update, and manage information in databases using a *structured query language (SQL)*, often embedded in a client application. These end-users of **OnLine** might be completely unaware that they are using **OnLine**. To them, **OnLine** is a nameless aspect of the system being used.

Application Developers

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

The concepts of relational databases managed by Informix database servers are explained in the *Informix Guide to SQL: Tutorial*. Other volumes, the *Informix Guide to SQL: Reference* and *Informix Guide to SQL: Syntax* provide invaluable information useful to application developers.

Database Administrators

The *database administrator* (DBA) of a database is primarily responsible for managing access control for a database as described in “Database Server Security” on page 1-8. 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 resource-intensive activities. The *Informix Guide to SQL: Tutorial* and *Informix Guide to SQL: Syntax* are also of interest to the DBA.

OnLine Administrators

Unlike the DBA, an **OnLine** administrator is responsible for maintenance, administration, and operation of the entire **OnLine** database server, which might be managing many individual databases. The tasks involved in **OnLine** administration are described in Chapter 2, “Overview of OnLine Administration.”

OnLine Operators

OnLine operators are responsible for carrying out routine tasks associated with **OnLine** administration such as backing up and restoring databases. The same person might fill the roles of the administrator and the operator.

Features Beyond the Scope of OnLine

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

No Bad-Sector Mapping

OnLine relies on the operating system of your host computer for bad-sector mapping. **OnLine** learns of a bad sector or a bad track when it receives a failure return code from a system call. When this happens, **OnLine** retries the access several times to ensure that the condition is not spurious. If the condition is confirmed, **OnLine** marks as *down* the chunk where the read or write was attempted.

OnLine 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 was attempted.

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

No Blob Scanning or Compression

OnLine receives blob data into an existing table in the following ways:

- From the **DB-Access** LOAD statement
- From the **dbload** utility
- From **INFORMIX-ESQL/C** locator variables
- From **INFORMIX-ESQL/COBOL** or **INFORMIX-ESQL/FORTRAN** FILE host data types

OnLine does not contain any mechanisms for scanning blobs and inserting the data into a file, or for blob compression, after the blob has been scanned.

Overview of OnLine Administration

Chapter Overview 3

Initial Tasks 3

Routine Tasks 3

 Changing Modes 4

 Archiving Data and Backing Up Logical-Log Files 4

 Monitoring OnLine Activity 4

 Checking for Consistency 4

Configuration Tasks 4

 Managing OnLine Instances 5

 Managing Database Logging Status 5

 Logical-Log Administration 5

 Physical-Log Administration 5

 Using Auditing 6

 Using Mirroring 6

 Using Data Replication 6

 Managing Shared Memory 6

 Managing Virtual Processors 7

Chapter Overview

As an **INFORMIX-OnLine Dynamic Server** administrator, you need to be aware of the tasks and responsibilities that fall into your domain. At first glance, the tasks might appear overwhelming. But, 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 **OnLine** 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 **OnLine**, you need to perform some initial installation and configuration tasks. These tasks are described in Chapter 3, “Installing and Configuring OnLine.”

If you are moving from one release level of **OnLine** to another release level, refer to the *INFORMIX-OnLine Dynamic Server Migration Guide*.

You must also configure connectivity for your database server and client applications, as explained in Chapter 4, “Configuring Connectivity.”

These tasks can seem complicated and time-consuming. Fortunately, they are not common tasks, and not representative of most of the administrative work **OnLine** 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

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

Changing Modes

The **OnLine** administrator is responsible for starting up and shutting down **OnLine** by changing the mode.

Chapter 7, “What Are **OnLine** Operating Modes?,” describes each **OnLine** mode and Chapter 8, “Managing Modes,” explains how to move **OnLine** from one mode to another.

Archiving Data and Backing Up Logical-Log Files

Frequent archiving of data and backing-up of logical-log files ensures that **OnLine** can be recovered in case of a failure. The *INFORMIX-OnLine Archive and Backup Guide* provides you with advice and guidelines for scheduling and coordinating archive activity with other tasks.

Monitoring OnLine Activity

OnLine design enables you to monitor every aspect of operation. Chapter 29, “Monitoring **OnLine**,” 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” on page 2-4.

Checking for Consistency

Informix recommends that you perform occasional checks for data consistency. Chapter 27, “What Is Consistency Checking?,” describes these checks.

Configuration Tasks

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

Managing OnLine Instances

If you plan to use more than one **OnLine** instance on the same computer, be aware of the issues explained in Chapter 11, “Managing Disk Space.”

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

Chapter 10, “Where Is Data Stored?,” explains the advantages and drawbacks of different disk configurations. Chapter 11, “Managing Disk Space,” describes the actual disk-management tasks.

Managing Database Logging Status

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

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

Logical-Log Administration

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

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

Information on backing up logical logs is in the *INFORMIX-OnLine Archive and Backup Guide*.

Physical-Log Administration

You can change the size and location of the physical log as part of effective disk management. See Chapter 21, “Managing the Physical Log.”

Using Auditing

If you use **OnLine** C2 level 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 they are abusing their access. These tasks, and others related to auditing, are explained in the *INFORMIX-OnLine Dynamic Server Trusted Facility Manual*.

Using Mirroring

Mirroring is described in Chapter 23, “What Is Mirroring?” If you plan to use mirroring (Informix recommends you mirror at least your root dbspace), instructions for using it are in Chapter 24, “Using Mirroring”.

Using Data Replication

Data replication uses additional hardware to provide a very high degree of availability. Data replication is described in Chapter 25, “What Is Data Replication?” If you plan to use data replication, also see Chapter 26, “Using Data Replication.”

Managing Shared Memory

Managing the use of shared memory is a broad task that falls under the responsibility of the **OnLine** administrator. Use of shared memory is described in Chapter 14, “OnLine Shared Memory.”

When managing memory you might do 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 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 15, “Managing OnLine Shared Memory,” describes the procedures to manage shared memory.

Managing Virtual Processors

The number and type of virtual processors that allows your **OnLine** database server to perform optimally depends on your hardware and on the type of database activity your database server supports.

Chapter 12, “What Is the Dynamic Scalable Architecture?,” explains what virtual processors are, and Chapter 13, “Managing Virtual Processors,” explains how to change the virtual-processor configuration.



Configuration





Installing and Configuring OnLine

Chapter Overview	3
Planning for INFORMIX-OnLine Dynamic Server	4
Consider Your Priorities	4
Consider Your Resources	4
Administering OnLine	5
Installing INFORMIX-OnLine Dynamic Server	6
Installing OnLine When No Other Informix Products Are Present	6
Installing OnLine When Other Informix Products Are Present	6
Installing OnLine When SE Is Already Present	6
Upgrading an Earlier Version of OnLine	7
Configuration Overview	8
Configuration Files	8
The onconfig.std File	8
The sqlhosts File	8
Environment Variables Used by OnLine	9
Multiple OnLine Database Servers	10
Configuring a Learning Environment	10
Log in as User informix	11
Choose Names	11
Set Environment Variables	11
Allocate Disk Space for Data Storage	12
Prepare the Cooked File Space	13
Prepare the ONCONFIG Configuration File	13
Preparing the ONCONFIG File for a Learning Environment	14

Prepare the Connectivity File	15
Preparing the sqlhosts File for the Learning Environment	15
Start OnLine Running	16
Practice Using OnLine	17
Configuring a Production Environment	17
Set Environment Variables	18
Prepare the ONCONFIG Configuration File	18
Overview of Configuration Parameters	19
Root Dbspace	20
Identification Parameters	20
Mirroring	21
Logical Logging	21
Physical Logging	22
Archiving and Logical-Log Backups	23
Message Files	23
Shared-Memory Parameters	24
Time Intervals in a Networked Environment	25
Data Migration	26
Allocate Disk Space	26
Prepare the Connectivity File	26
Prepare the ON-Archive Configuration File	26
Prepare for Native Language Support	27
Evaluate UNIX Kernel Parameters	27
Start OnLine and Initialize Disk Space	27
Create Blobspaces and Dbspaces	28
Do Administrative Tasks	28
Prepare UNIX Startup and Shutdown Scripts	28
Warn UNIX System Administrator About cron Jobs	29
Make Arrangements for Tape Management	30
Make Sure Users Have the Correct Environment Variables	30

Chapter Overview

Implementing an **INFORMIX-OnLine Dynamic 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 **OnLine** can greatly affect the performance of database operations. You can customize **OnLine** so that it functions optimally in your particular data processing environment. For example, an **OnLine** instance serving 1,000 users who execute frequent, short transactions is quite different from the **OnLine** instance where a few users make long and complicated searches.

This chapter has two purposes: to let you quickly start an **OnLine** database server using a simple configuration, and to provide a point of orientation for a more studied configuration process. It also discusses some of the issues you must consider before installing **OnLine**, and introduces terminology. The following topics are in this chapter:

- Planning for **OnLine**
- Installing **OnLine**
- Configuration overview
- Configuring **OnLine** for a learning environment
- Configuring **OnLine** for a production environment

The sections on installation include special instructions for installing Version 6.0 **OnLine** when other Informix products are already installed on your computer.

The sections on the configuration files will 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 your **OnLine** database server. This chapter gives pointers to more detailed discussions in the rest of the book.

Planning for INFORMIX-OnLine Dynamic Server

When planning for **OnLine**, 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, you need to keep in mind the characteristics of your database server, such as:

- What is your highest priority, transaction speed or safety of the data?
- Will the database server usually handle short transactions or fewer long transactions?
- Will this **OnLine** instance be used by applications on other computers?
- What is the maximum number of users you can expect?
- How much help or supervision will the users require? To what extent do you want to control the environment of the users?
- Are you limited by resources for space? CPU? Availability of operators?
- How much does the **OnLine** instance have to do without supervision?

Consider Your Resources

Before you start the initial configuration, you should 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 **OnLine**, everything—tables, log files, indexes, data—goes into the root dbspace on one disk drive. After **OnLine** 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 11, “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. **OnLine** keeps statistics that help you adjust these parameters after your **OnLine** database server

has been running for a while. Your archiving strategy also needs to take into account availability of tape drives. Archiving is discussed in the *INFORMIX-OnLine Dynamic 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 **OnLine**?

You might need the assistance of the UNIX system administrator to evaluate and modify the UNIX kernel parameters.

- What are the network names and addresses of the other computers on your network? Does your system run NIS?

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

Administering OnLine

OnLine provides a variety of administrative tools to create configuration files, to change modes, to change various aspects of your configuration, and to monitor statistics about the database server. The tools include:

- ON-Monitor
- ON-Archive
- ON-Audit
- **OnLine** utilities

These tools are discussed in Chapter 34, “ON-Monitor,” and Chapter 37, “OnLine Utilities.”

Other tools used for monitoring the behavior of the **OnLine** database server include:

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

These tools are discussed in Chapter 38, “OnLine Message Log Messages,” and Chapter 36, “The sysmaster Database.”

Installing INFORMIX-OnLine Dynamic Server

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

- Installing **OnLine** when no other Informix products are present
- Upgrading a previous version of **OnLine**
- Installing **OnLine** when **INFORMIX-SE** is already present
- Running multiple **OnLine** database servers on one host computer

Installing OnLine When No Other Informix Products Are Present

The *UNIX Products Installation Guide* gives complete instructions for installing **OnLine** when no other Informix products are already installed on your computer. Please refer to the *UNIX Products Installation Guide* for instructions.

Installing OnLine When Other Informix Products Are Present

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

1. Client application products, such as **INFORMIX-ESQL/C** or **INFORMIX-4GL**
2. Database server products, such as **OnLine** and **INFORMIX-SE**
3. Networking products, such as **INFORMIX-STAR** or **INFORMIX-NET**
4. Gateway products, such as **INFORMIX-Gateway with DRDA**

The *UNIX Products Installation Guide* gives instructions for installing each product. The individual product guides give any additional information that might be required. Please refer to those guides.

Installing OnLine When SE Is Already Present

If you are installing **OnLine** on a computer that already has an **INFORMIX-SE** database server, you do not need to create a new **\$INFORMIXDIR** directory. **OnLine** can be installed in the same directory as **INFORMIX-SE**. Follow the

installation instructions in the *UNIX Products Installation Guide* but skip the steps that create group **informix**, user **informix**, and the **informix** directory because they should already exist.

After **OnLine** is installed, configured, and working properly, you might want to move databases from **INFORMIX-SE** to **OnLine**. **OnLine** databases and **SE** databases have different internal formats. To move databases from **SE** to **OnLine**, refer to “Moving Data from SE to OnLine” on page 31-16.

Upgrading an Earlier Version of OnLine

When you install **INFORMIX-OnLine Dynamic Server**, Version 6.0, on a computer that already has an earlier version of **OnLine**, you must do the following tasks before installation:

1. Decide where to store the Version 6.0 files.

When you install **OnLine** it overwrites any files associated with **OnLine** that may exist in the **\$INFORMIXDIR** directory. If you want to preserve your files of previous versions files, you should create a new directory for the Version 6.0 product.

2. Make sure you have enough space for the new **OnLine**.

Refer to the *INFORMIX-OnLine Dynamic Server Migration Guide* for information about space requirements.

3. Protect your current (pre-6.0) data.

To protect your current data, you should run checks for consistency and make a level-0 archive. Refer to the archiving instructions in the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for your current version of **OnLine**.

4. Save a copy of the current (pre-6.0) 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 the new software will be installed. Environment variables are discussed in Chapter 4 of the *Informix Guide to SQL: Reference*.
7. Follow the instructions in the *UNIX Products Installation Guide* but do not re-create the **informix** user and directory. The script installs **OnLine**, Version 6.0, into the **\$INFORMIXDIR** directory specified for user **root**. Note that the installation script does *not* bring **OnLine** on-line.
8. Refer to the *INFORMIX-OnLine Dynamic Server Migration Guide* for upgrade instructions.



Warning: Do not initialize disk space when you upgrade your **OnLine**. If you initialize disk space, you will destroy all of your existing data.

Configuration Overview

After **OnLine** is installed, it must be configured before it can be brought on-line. *Configuration* refers to setting specific parameters that customize the **OnLine** 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 used by **OnLine**. It provides a foundation for more detailed information about configuration that is discussed throughout this book.

Configuration Files

A complete summary of the files used by **OnLine** is in Chapter 42, “Files Used by **OnLine**.” The two files that are used in configuring all **OnLine** 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 **OnLine** installation procedure. The **onconfig.std** file contains default values 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 42-10.

The *sqlhosts* File

The `$INFORMIXDIR/etc/sqlhosts` file is the *connectivity file*. It contains information that enables an Informix client application to connect to any Informix database server on the network. It specifies the database server name, the type of connection, the name of the host computer, and the service name.

You must prepare the **sqlhosts** file even if both the client application and the **OnLine** database server are on the same computer. The **sqlhosts** file is covered in detail in “The `$INFORMIXDIR/etc/sqlhosts` File” on page 4-10.

Environment Variables Used by OnLine

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 of the following environment variables, which must be correctly set before you can initialize **OnLine**:

- 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 **OnLine** 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, **OnLine** uses configuration values from the file `$INFORMIXDIR/etc/onconfig`. For information about the **onconfig** file, see “onconfig” on page 42-7.

The INFORMIXSERVER environment variable specifies the name of the default database server. You set it after you prepare the ONCONFIG configuration file. Strictly speaking, INFORMIXSERVER is not required for initialization. However, if INFORMIXSERVER is not set, **OnLine** does not build the **sysmaster** tables. (Refer to “What Is the sysmaster Database?” on page 36-3.) 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 **OnLine** 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.

Multiple OnLine Database Servers

When more than one independent **OnLine** 6.0 database server runs on the same host computer, it is called *multiple residency*. To prepare to use multiple **OnLine** database servers, first install and configure *one* **OnLine** database server following the instructions in this chapter. Then refer to Chapter 6, “Using Multiple Residency.”



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

Configuring a Learning Environment

If this is your first experience with **OnLine**, you might want to start by configuring a learning environment. The learning environment allows you to prepare a working **OnLine** database server with a minimum of time and effort. It is also a quick way to check that the new **OnLine** is working correctly, that is, that the installation was successful.

The instructions in this section allow you to build an **OnLine** database server that is suitable for a few users and moderate-sized databases. This environment is not expected to serve as a final configuration; it just allows you to see a working **OnLine** database server. After you have some practice using **OnLine**, you can reconfigure the database server for a production environment. (See “Configuring a Production Environment” on page 3-17.)

The following list outlines the steps for creating a learning environment. Each step is described in detail in the following sections.

1. Log in as user informix
2. Choose names for your configuration file and your database server
3. Set environment variables
4. Allocate disk space for data storage
5. Prepare an ONCONFIG configuration file
6. Prepare the connectivity file (**sqlhosts**)
7. Start **OnLine**
8. Practice using **OnLine**

Log in as User *informix*

Most administrative tasks require that you log in as user **informix** or user **root**. For this example, you should log in as user **informix**.

Choose Names

Choose a name for your ONCONFIG configuration file that indicates how the file is used. The examples in this section use the name **onconfig.learn** for the ONCONFIG file.

Choose a name for your **OnLine** database server. This name is called the *dbservername*. You will use the *dbservername* when you set environment variables, in the ONCONFIG configuration file, and in the *sqlhosts* file. The examples in this section use the name **learn_online**.

The *dbservername* must be 18 or fewer characters and can include lowercase characters, numbers, and underscores. It should begin with a letter. The database server name must be unique within your network, so Informix recommends choosing a descriptive name such as **online_hostname** or **accounting_online1**.

Set Environment Variables

Before you start the configuration process, set the following environment variables:

- Set the **INFORMIXDIR** environment variable to the full pathname of the directory in which **OnLine** is installed.
- Set the **PATH** environment variable to include the **\$INFORMIXDIR/bin** directory.
- Set the **ONCONFIG** environment variable to the configuration file that you just chose: **onconfig.learn**.
- Set the **INFORMIXSERVER** environment variable to the database server name that you just chose: **learn_online**.

For information about setting environment variables, refer to your UNIX documentation. The following example illustrates setting the **ONCONFIG** environment variable in the C shell and the Bourne (or Korn) shell:

```
C shell:          setenv ONCONFIG onconfig.learn
Bourne shell:   ONCONFIG=onconfig.learn
                  export ONCONFIG
```

Allocate Disk Space for Data Storage

The UNIX operating system allows you to use two different types of disk space: raw and cooked. *Cooked disk space* or *cooked file space* refers to ordinary UNIX files. It is space that has already been organized and that UNIX administrators for you. *Raw disk space* is unformatted space that **OnLine** administrators. **OnLine** allows you to use either type of disk space (or a mixture of both types).

To gain the full benefits of **OnLine** capabilities, you must use raw space. However, cooked space is easier to use and is acceptable for many environments. The instructions for the learning environment assume that you will use cooked space. To learn more about cooked space and raw space, refer to “Where Is Data Stored?” in Chapter 10 for a general discussion and to “Managing Disk Space” in Chapter 11 for instructions about allocating disk space.

Informix refers to its biggest unit of physical disk storage as a *chunk*. For your learning environment, the file you create becomes one chunk. Later, in a production environment, you will allocate more chunks. For more information about chunks, refer to Chapter 11, “Managing Disk Space.”

The cooked file for your learning environment should be in a directory that you control and that has sufficient space allocated. The default values given in the configuration file require 20 megabytes of disk space. Do not put your cooked file space in the home directory of user **informix**, nor in the directory where the executable code for the **OnLine** database server is installed (the `$INFORMIXDIR`). Space requirement issues are discussed in “How Much Disk Space Do You Need to Store Your Data?” on page 10-27.

Note: The default values in the configuration file assume that you are starting an active, medium-sized production system. If your learning environment is short of space, you can reduce the default value of `ROOTSIZE` from 20 megabytes to 7 megabytes, without changing other parameters in your `ONCONFIG` file.

Prepare the Cooked File Space

The details for setting up a cooked file space vary slightly from one UNIX installation to another. Figure 3-1 shows a typical example of the commands you need to prepare the cooked disk space. This example assumes that you plan to store the cooked space in the file: `/usr/data/root_chunk`.

Step	Command	Comments
1.	<code>% cd /usr/data</code>	Change directories to the directory where the cooked space will reside.
2.	<code>% cat /dev/null > root_chunk</code>	Create your root chunk by concatenating null to a file. Informix recommends that you name this file something descriptive, such as <code>root_chunk</code> , to simplify keeping track of your space.
3.	<code>% chmod 660 root_chunk</code>	Set the permissions of the file to 660 (rw-rw----).
4.	<code>% ls -lg root_chunk</code> <pre>-rw-rw---- 1 informix informix 0 Oct 12 13:43</pre>	Verify that both group and owner of the file are informix. You should see something similar to this line (which is wrapped around in this example).

Figure 3-1 Preparing cooked file space for OnLine

Prepare the ONCONFIG Configuration File

The ONCONFIG *configuration file* contains values for parameters that describe the OnLine environment. You can have several different ONCONFIG configuration files to describe different environments, such as learning, development, and production.

An overview of the ONCONFIG configuration parameters is in “Overview of Configuration Parameters” on page 3-19. Chapter 35, “OnLine Configuration Parameters,” gives a complete list of the ONCONFIG configuration parameters, including a summary of their functions and default values.

All OnLine configuration files reside in the `$INFORMIXDIR/etc` directory. One of the files loaded during the installation of OnLine is `onconfig.std` (“OnLine configuration standard”). It contains default values for the ONCONFIG parameters and serves as the template for all other ONCONFIG configuration files that you create. *Do not modify `onconfig.std`!*

To prepare a configuration file for the learning environment, copy `onconfig.-std` into your own configuration file and then modify the parameters. Informix provides a menu-based utility, ON-Monitor, for modifying the configuration file. You can use ON-Monitor to modify your configuration file,

but because only a few values need to be set for the learning environment, it is probably more convenient to use a text editor, such as **vi** or **emacs**. The ON-Monitor utility is described in Chapter 34, “ON-Monitor.”

Preparing the ONCONFIG File for a Learning Environment

To prepare the ONCONFIG file for a learning environment, you should follow these steps:

1. Change directories to the `$INFORMIXDIR/etc` directory and copy the `onconfig.std` file into a new file in that directory, using the name that you chose for your ONCONFIG file. For example:

```
cd $INFORMIXDIR/etc
cp onconfig.std onconfig.learn
```

You can use `$INFORMIXDIR` as a variable so that you do not need to keep typing a (possibly) long pathname.

2. Edit `onconfig.learn` to modify these parameters:

- `ROOTPATH` **`/usr/data/root_chunk`**

`ROOTPATH` is the full directory pathname of the cooked file space that you created in the previous section. In this example, the pathname of the cooked space is `/usr/data/root_chunk`.

- `TAPEDEV` **`/dev/null`**
 `LTAPEDEV` **`/dev/null`**

Setting these parameters to `/dev/null` allows **OnLine** to behave as if tape drives were present and log files were being backed up, but in fact the output to tape is discarded. With these settings, you cannot restore data. Also, ON-Archive does not work if `LTAPEDEV` is set to `/dev/null`.

For a production environment, or to gain experience working with archive and backup tools, you should set `TAPEDEV` and `LTAPEDEV` to actual devices. For more information about logging, refer to Chapter 16, “What Is Logging?” Refer to the *INFORMIX-OnLine*

Dynamic Server Archive and Backup Guide for discussions of the effects of `/dev/null`.

- `DBSERVERNAME` `learn_online`

Change the `DBSERVERNAME` parameter to the `dbservername` you chose for your database server. For this example, we have used `learn_online`.

You already used the `dbservername` when you set the `INFORMIXSERVER` environment variable. You will use the `dbservername` in the next section when you prepare the `sqlhosts` file. `INFORMIXSERVER` tells an application which database server to use, and `sqlhosts` tells the application how to connect to the database server.

- `SERVERNUM` `some_number`

Change the `SERVERNUM` parameter to some integer between 0 and 255. Each different **OnLine** instance must have a distinct value. You could leave `SERVERNUM` set to its default value 0 for your first **OnLine**, but that can cause problems in **ON-Monitor** when you start to initialize a second **OnLine**. It is safer to set `SERVERNUM` to a unique, nonzero value each time you make a new `ONCONFIG` file.

- `MSGPATH` `a_pathname`

If you installed the **OnLine** executables in the `/usr/informix` directory, as suggested in the *UNIX Products Installation Guide*, you do not need to change this parameter. Otherwise set it to the directory where you want the message log to be stored.

Prepare the Connectivity File

The `sqlhosts` file contains information that allows a client application to connect to a database server. For the learning environment, you can use the simplest possible connection, using shared memory.

For more information about shared-memory connections, refer to “The Communications Portion of OnLine Shared Memory” on page 14-29. For information about `sqlhosts` beyond what is needed for a simple configuration, refer to “The `$INFORMIXDIR/etc/sqlhosts` File” on page 4-10.

Preparing the `sqlhosts` File for the Learning Environment

Make sure you are in the `$INFORMIXDIR/etc` directory and edit the `sqlhosts` file. (The `sqlhosts` file should already be present. If it is not, create it.)

You need to add one line (one entry) to the **sqlhosts** file. The entry has the following four fields:

1. The **dbservername**. You have already used the **dbservername** to set the **INFORMIXSERVER** environment variable and as the value of the **DBSERVERNAME** parameter. This example uses **learn_online**.
2. The type of connection. For a shared-memory connection, the value needed is **onipcshm**.
3. The name of your host computer.
4. The **servicename**. For a shared-memory connection, the following statements are true:
 - The **servicename** can be any short group of letters and numbers. This example uses the value **xyz**.
 - The **servicename** does not need to appear in any of the network connectivity files. (We will discuss these files later, when we discuss network connections.)

You can separate the fields in the **sqlhosts** file with spaces or tabs. If the host name of your computer is **myhost**, the **sqlhosts** entry looks like this:

```
learn_online onipcshm myhost xyz
```

Start OnLine Running

To start **OnLine** running for the first time, you must initialize both the disk space and the shared memory that is used by the **OnLine** database server. To do this, execute the following command:



Warning: When you execute this command, all existing data in the **OnLine** disk space is destroyed. The **-i** flag is used **ONLY** when you are starting a brand-new **OnLine**.

```
% oninit -i
```

If you want to stop **OnLine**, execute the following command:

```
% onmode -k
```

This command asks if you really want to take **OnLine** off-line. It then tells you how many users are currently using **OnLine** and asks if you want to proceed. If you answer “y” to both questions, **OnLine** comes to off-line mode.

If you stop **OnLine** and want to restart it without destroying the information on disk, use this command:

```
% oninit
```

You can also start or stop **OnLine** using the ON-Monitor Mode menu. Refer to “ON-Monitor Screen Options” on page 34-4.

For more information on initialization, see Chapter 9, “What Is Initialization?” The various ways you can start the **OnLine** database server are discussed in “oninit: Initialize OnLine” on page 37-16.

Practice Using OnLine

Your **OnLine** product includes **DB-Access**, an application that allows you to create and query databases and tables. It is installed as part of the installation process for **OnLine**. **DB-Access** includes scripts to create a practice database that you can use to try out various **OnLine** features. Refer to the *DB-Access User Manual* for information about creating the practice database.

You might want to practice using an archive and backup tool at this point. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for guidance.

Configuring a Production Environment

The previous section, “Configuring a Learning Environment,” covered the minimum steps required to start a very basic **OnLine** database server. This section covers the steps that you take to prepare an **OnLine** database server for an actual production environment. It also gives pointers into other chapters that include detailed discussions of specific topics.

The following initial configuration tasks are discussed in this chapter:

1. Set environment variables
2. Prepare the ONCONFIG configuration file
3. Allocate disk space
4. Prepare the connectivity configuration file

5. Prepare the ON-Archive configuration file
6. Prepare for native language support
7. Evaluate UNIX kernel parameters
8. Start **OnLine** and initialize disk space
9. Create blobspaces and dbspaces (if desired)
10. Do administrative tasks

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 **OnLine** 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`).

The Native Language Support (NLS) feature of **OnLine** lets you use non-English characters, monetary conventions and/or collating sequences. NLS is discussed in Chapter 1 of the *Informix Guide to SQL: Reference*. If you are using the NLS features of **OnLine**, you need to set following the NLS environment variables:

- `DBNLS`
- `LANG`
- `LC_xxx`

Prepare the ONCONFIG Configuration File

You can create and modify the `ONCONFIG` configuration file using a standard editor or by using the `ON-Monitor` utility. Directions for using `ON-Monitor` are in Chapter 34, “`ON-Monitor`.”

To prepare the `ONCONFIG` configuration file using a standard 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 **OnLine**, you can leave most of the parameters set to their default values.

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

- ROOTPATH page 35-36
- SERVERNUM page 35-37
- DBSERVERNAME page 35-12

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

- DBSERVERALIASES page 35-11

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

- TAPEDEV page 35-42
- LTAPEDEV page 35-24

You should verify that the following parameters have valid pathnames:

- MSGPATH page 35-27
- CONSOLE page 35-11

After your **OnLine** database server is configured and running, you should review the configuration parameters. "Monitoring Configuration Information" on page 29-9 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 pointers 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 **OnLine** database server. The parameters that describe the root dbspace are as follows:

- ROOTNAME page 35-35
- ROOTPATH page 35-36
- ROOTOFFSET page 35-35
- ROOTSIZE page 35-36

You can choose any descriptive name for the ROOTNAME, but it is usually called **rootdbs**, which is its default value. The ROOTPATH is the pathname of the storage allocated to the root dbspace. Choosing and allocating of the storage are discussed in Chapter 11, “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 10-27. “Do You Need to Specify an Offset?” on page 11-5, discusses the circumstances when you need to set ROOTOFFSET.

Identification Parameters

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

- DBSERVERNAME page 35-12
- DBSERVERALIASES page 35-11
- SERVERNUM page 35-37

DBSERVERNAME specifies the name of the **OnLine** database server. The name specified in DBSERVERNAME is called the *database server name* or *dbserver-name*. 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 4-17.

DBSERVERALIASES specifies a list of alternative dbservernames when multiple communication protocols are used. (See “The DBSERVERALIASES Configuration Parameter” on page 4-18.)

The value given by `SERVERNUM` specifies a unique ID corresponding to the **OnLine** instance. It must be unique for each **OnLine** database server on your local host but does not need to be unique across your network. `SERVERNUM` is described in “The Role of the `SERVERNUM` Configuration Parameter” on page 5-4.

Mirroring

Mirroring allows very fast recovery from a disk crash while **OnLine** 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. These parameters describe mirroring of the root dbspace:

- `MIRROR` page 35-26
- `MIRRORPATH` page 35-26
- `MIRROROFFSET` page 35-26

Mirroring has both positive and negative effects on performance: disk updates are slower; disk reads are faster. Mirroring is discussed in Chapter 23, “What Is Mirroring?”

Logical Logging

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

- `LOGBUFF` page 35-20
- `LOGFILES` page 35-20
- `LOGSIZE` page 35-21
- `LOGSMAX` page 35-21
- `LTXHWM` page 35-25
- `LTXEHWM` page 35-25

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 default value for `LOGBUFF` unless your database server has an unusually large number of transactions. After your **OnLine** database server is operative, you use the system statistics to tune the buffer size. `LOGBUFF` is discussed in “Logical-Log Buffer” on page 14-24 and is described in “`LOGBUFF`” on page 35-20.

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. See “What Should Be the Size and Number of Logical-Log Files?” on page 18-7.

A logical-log file cannot be reused by **OnLine** unless all of the transactions recorded in the log have been completed. Because of this, a transaction that takes a long time, a *long transaction*, can cause performance problems. The *long -transaction high-water mark* parameter, LTXHWM, specifies the percentage of the available logical log that can be used before **OnLine** takes moderate action to avoid the undesirable effects of reaching the point of LTEXHWM. A companion parameter, the *long-transaction exclusive-access high-water mark*, LTEXHWM, is the point at which **OnLine** takes drastic action. These parameters are described in “Avoiding Long Transactions” on page 18-12.

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. These parameters describe the physical log:

- PHYSDBS page 35-33
- PHYSDBS page 35-33
- PHYSBUFF page 35-32

PHYSDBS specifies the size of the physical log. PHYSDBS specifies the name of the dbspace where the physical log resides. When **OnLine** 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 PHYSDBS or PHYSDBS, you must make a level-0 archive. These parameters are discussed in Chapter 20, “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 14-25, and is described in “PHYSBUFF” on page 35-32.

Archiving and Logical-Log Backups

You can create archives and make logical-log backups for data managed by **OnLine** using one of the following tools:

- ON-Archive
- **ontape**

You must choose one tool or the other. ON-Archive is the more powerful and flexible method. It does not use parameters from the ONCONFIG file. Instead it uses configuration parameters in the **config.arc** configuration file, described in the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*.

The other method for archiving and logical-log backups uses the **ontape** utility. It is easier to configure but does not give as much flexibility or power. 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-Archive). These parameters are as follows:

- TAPEDEV page 35-42
- TAPEBLK page 35-41
- TAPESIZE page 35-44
- LTAPEDEV page 35-24
- LTAPEBLK page 35-23
- LTAPESIZE page 35-24

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-OnLine Dynamic Server Archive and Backup Guide*.

Message Files

The message files provide information about how **OnLine** is functioning. These parameters give the pathnames of the message files:

- CONSOLE page 35-11
- MSGPATH page 35-27

CONSOLE specifies the pathname for console messages. Messages that **OnLine** directs to the console require immediate action. The default value, **/dev/console**, sends messages to the system console screen. MSGPATH is the pathname of the **OnLine** message file. **OnLine** writes status messages and

diagnostic messages to this file. You should monitor this file regularly. The message log is described in “What Is the Message Log?” on page 29-6. The messages themselves are discussed in Chapter 38, “OnLine Message Log Messages.”

Shared-Memory Parameters

The following shared-memory parameters are very important to the performance of **OnLine**. They describe how space is allocated in shared memory.

- USERTHREADS page 35-46
- BUFFERS page 35-8
- LOCKS page 35-19
- TRANSACTIONS page 35-45
- SHMBASE page 35-38
- CHUNKS page 35-9
- DBSPACES page 35-13
- TBLSPACES page 35-44
- RESIDENT page 35-35
- LRUS page 35-22
- LRU_MAX_DIRTY page 35-22
- LRU_MIN_DIRTY page 35-23
- CKPTINTVL page 35-10

USERTHREADS is the maximum number of user processes that can concurrently attach to shared memory. You need to set USERTHREADS carefully because the value you choose determines minimum values for three other parameters: BUFFERS, LOCKS, and TRANSACTIONS.

BUFFERS is the number of shared-memory buffers available to the **OnLine** database server. See “Shared-Memory Buffer Pool” on page 14-23 for a discussion of shared-memory buffers.

LOCKS specifies the maximum number of locks available to **OnLine** user processes during transaction processing. You can use the default value (page 35-19) for LOCKS.

TRANSACTIONS is the maximum number of concurrent transactions supported by **OnLine**. It is automatically set to the value of USERTHREADS. You should not change the value unless you plan to use an X/Open environment. (Refer to the *INFORMIX-TP/XA User Manual*.)

SHMBASE is the shared-memory base address and is machine-dependent. Its value is usually not changed.

CHUNKS specifies the maximum number of chunks supported by **OnLine**. The value should be as close as possible to the maximum number of chunks permitted, which is system dependent.

DBSPACES specifies the maximum number of dbspaces and is equal to or less than the value of CHUNKS because each dbspace requires at least one chunk.

TBLSPACES is the maximum number of active tblspaces. The total of all disk space allocated to a table is that table's tblspace, including data, indexes, blob data, and the bit-map pages that track page usage.

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 15-3.

The LRUS (Least Recently Used) queues manage the shared-memory pool of pages (memory spaces) where all activity of the database takes place. The LRU parameters are used for performance tuning. You should use the default values for the initial configuration. These parameters are discussed in "OnLine LRU Queues" on page 14-32.

CKPTINTVL, the checkpoint interval, is the maximum time allowed to elapse before a checkpoint. You should use the default values for the initial configuration. Modifying CKPTINTVL to improve performance is discussed in "Setting Configuration Parameters for the Shared-Memory Performance Options Using ON-Monitor" on page 15-12.

Time Intervals in a Networked Environment

- DEADLOCK_TIMEOUT page 35-14
- TXTIMEOUT page 35-45

DEADLOCK_TIMEOUT specifies the amount of time that **OnLine** waits for a shared-memory resource during a distributed transaction. TXTIMEOUT is the amount of time a participant **OnLine** 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 32-34.

Data Migration

Chapter 31, “Data Migration,” explains the different options available for moving data to or from **OnLine**. If, after looking at that information, you decide to use **onload** and **onunload** to migrate data, you might want to set one of the following parameters:

- TAPEDEV page 35-42
- LTAPEDEV page 35-24

See “onunload: Transfer Binary Data in Page Units” on page 37-77 and “onload: Create a Database or Table” on page 37-18.

Allocate Disk Space

Before you allocate the disk space, you should study the information about disk space in Chapter 10, “Where Is Data Stored?” Directions for allocating raw disk space are given in that chapter. If you want to use cooked disk space, you can follow the instructions from “Prepare the Cooked File Space” on page 3-13.

Prepare the Connectivity File

The connectivity file, `$INFORMIXDIR/etc/sqlhosts`, contains information that is required to allow an Informix client application to connect to an Informix database server, in this case **OnLine**. The content of the `sqlhosts` file is covered in “The `$INFORMIXDIR/etc/sqlhosts` File” on page 4-10.

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

If you are using a shared-memory interface on a single computer, you can use the very simple `sqlhosts` entry illustrated in “Prepare the ONCONFIG Configuration File” on page 3-13.

Prepare the ON-Archive Configuration File

ON-Archive uses several configuration files, as described in “Chapter Overview” on page 42-3. The most important file when configuring ON-Archive is the `config.arc` file. It describes, among other things, the devices used for different archiving and backup tasks. The *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* describes the parameters for the `config.arc` file.

Prepare for Native Language Support

The NLS feature of all Version 6.0 products allows you to create databases using the diacritics, collating sequence, and monetary and time conventions of the selected language. There are no ONCONFIG configuration parameters for NLS, but you must set the appropriate environment variables. NLS is discussed in Chapter 1 of the *Informix Guide to SQL: Reference*.

Evaluate UNIX Kernel Parameters

Your OnLine product arrives with a machine-specific file called `$(INFORMIXDIR)/release/ONLINE_6.0`, which contains recommended values for UNIX kernel parameters. Compare the values in this file with your current UNIX configurations.

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

If the recommended values for OnLine differ significantly from your current environment, consider modifying your UNIX kernel settings. Background information that describes the role of the UNIX kernel parameters in OnLine is in “UNIX Kernel Configuration Parameters” on page 15-3.

Start OnLine and Initialize Disk Space

To bring OnLine to on-line mode, you can type the following command at the system prompt:

```
oninit
```

If you are starting a brand-new OnLine database server, use the following command to initialize the disk space as well as to bring OnLine into on-line mode:



Warning: When you execute this command, all existing data in the OnLine disk space is destroyed. The `-i` flag is used ONLY when you are starting a brand-new OnLine.

```
% oninit -i
```

You can also initialize disk space using ON-Monitor. (See Chapter 34, “ON-Monitor.”)

Create Blobspaces and Dbspaces

Now that **OnLine** is initialized, you can create blobspaces and dbspaces as desired. Blobspaces and dbspaces are described in Chapter 10, “Where Is Data Stored?” The allocation and management of blobspaces and dbspaces are discussed in Chapter 11, “Managing Disk Space.”

Do Administrative Tasks

After you initialize **OnLine**, you need to do 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 **OnLine** automatically when your computer enters multiuser mode. You can also modify your UNIX shutdown script to shut down **OnLine** in a controlled manner whenever UNIX shuts down.

Prepare the Startup Script

To prepare the UNIX startup script, add UNIX and **OnLine** 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 **OnLine** 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 the **sysmaster** database can be updated (or created, if needed).

- Execute **oninit**, which starts **OnLine** and leaves it in on-line mode.
If you plan to initialize multiple versions of **OnLine** (multiple residency), you must reset **ONCONFIG** and **INFORMIXSERVER** and re-execute **oninit** for each instance of **OnLine**.
- If you are using ON-Archive, you might want start **oncatlgr**. Refer to the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for more information.

If different versions of **OnLine**, such as 5.0 **OnLine** and 6.0 **OnLine**, are installed in different directories, you must reset **INFORMIXDIR** and repeat the preceding steps for each different version.

Prepare the Shutdown Script

To shut down **OnLine** in a controlled manner whenever UNIX shuts down, add UNIX and **OnLine** 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 **OnLine** 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 **OnLine** off-line.

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

If different versions of **OnLine**, such as 5.0 **OnLine** and 6.0 **OnLine**, are installed in different directories, you must reset **INFORMIXDIR** and repeat the preceding steps for each different version.

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

Warn UNIX System Administrator About *cron* Jobs

OnLine creates the **.inf.servicename** and/or **VP.servernameC** files in the **/tmp** directory. Some UNIX systems run cron jobs that routinely delete all files from the **/tmp** directory. For information about these files, see Chapter 42, "Files Used by OnLine."

Make Arrangements for Tape Management

When you plan your data archive and logical-log backup schedule, as discussed in the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*, you need to take into account the availability of tape devices to manage the data, and the availability of operators to perform the archives and 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

You need to make sure that every user of an Informix product has the correct environment variables. Environment variables are discussed in Chapter 4 of *Informix Guide to SQL: Reference*.

Each user must set the following environment variables before accessing **OnLine**:

- INFORMIXSERVER
- INFORMIXDIR
- PATH

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

There are three techniques 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 **OnLine** user so that these environment variables are set during login.
- Educate your users to set the environment variables by hand every time they want to work with **OnLine**.

The user might need other environment variables, such as TERMCAP and LC_COLLATE to fully describe his/her environment. You can prepare an environment configuration file, `$INFORMIXDIR/etc/informix.rc`, which sets additional environment variable for each user of **OnLine**. The individual user 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 alternate qualifier defaults file. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for more information about ON-Archive.

Configuring Connectivity

Chapter Overview	3
Types of Client/Server Connections	3
Shared-Memory Connections	4
Network Connections	5
Connection Between Two Computers	5
Local Loopback Connections	6
Connectivity Files	7
Network Configuration Files	7
Using the TCP/IP Communication Protocol	8
Using IPX/SPX Connections	9
Network Security Files	9
The <code>/etc/hosts.equiv</code> and <code>~/.rhosts</code> Files	9
The <code>~/.netrc</code> File	10
The <code>\$INFORMIXDIR/etc/sqlhosts</code> File	10
Editing the <code>sqlhosts</code> File	12
The <code>dbservername</code> Field	12
The <code>nettype</code> Field	12
The <code>hostname</code> Field	14
The <code>servicename</code> Field	15
ONCONFIG Parameters for Connectivity	17
The <code>DBSERVERNAME</code> Configuration Parameter	17
The <code>DBSERVERALIASES</code> Configuration Parameter	18
Environment Variables for Network Connections	19
Examples of Client/Server Configurations	19
Using a Shared-Memory Connection	20
Using a Local Loopback Connection	21

Using a Network Connection	21
Using Multiple Connection Types	23
Accessing Multiple 6.0 OnLine Database Servers	25
Using the 6.0 Relay Module	26
A Relay Module Configuration with Three Database Servers	28
Using 5.0 INFORMIX-STAR or 5.0 INFORMIX-NET	29
Using a 6.0 Client Application with a 5.0 Database Server	30

Chapter Overview

This chapter describes how to configure database server and client environments so that client applications can connect to **INFORMIX-OnLine Dynamic Server** database servers. The `$INFORMIXDIR/etc/sqlhosts` file contains the necessary information about the location and connection type(s) of each database server. The parameters in the ONCONFIG configuration file let you adjust the **OnLine** database server for different environments.

The chapter is divided into the following parts:

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

Note: For the remainder of this chapter, the `$INFORMIXDIR/etc/sqlhosts` file will be referred to as the `sqlhosts` file.

Types of Client/Server Connections

The connection between a client application and a database server is handled by functionality that is integrated into all Informix Version 6.0 database server products. This functionality handles all connections between the client application and the database server. To connect to a database server (using, for example, the CONNECT statement), the client application specifies only the name of the database server. **OnLine** 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.

Version 6.0 of **OnLine** introduces several changes to the way connections between client applications and database servers are handled. These changes are summarized in **Changes from Version 5.0** in the release notes and in the *INFORMIX-OnLine Dynamic Server Migration Guide*.

OnLine uses the following types of connections to communicate between client applications and database servers:

- Shared-memory connections
- Network connections

You can connect many different client applications to the same **OnLine** database server, using both shared memory and network connections. The following sections explain each of these connections.

Shared-Memory Connections

You can use *shared-memory* connections only when the client application and the database server are on the same computer. The shared memory used for communications is not the same as the resident shared memory used by **OnLine**. See “The Communications Portion of OnLine Shared Memory” on page 14-29 for a discussion of shared-memory communications. Figure 4-1 shows a shared-memory connection.

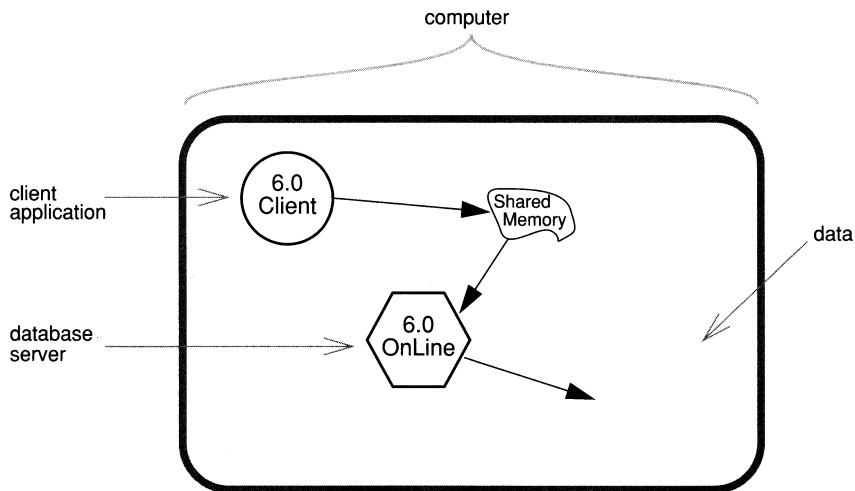


Figure 4-1 A shared-memory connection

Shared memory provides very fast access to a database server, but it poses some security risks. Shared-memory communication is vulnerable to programming errors if the client application does explicit memory addressing or over-indexes data arrays. Such errors do not affect the application if you use

IPC unnamed pipes or network communication. For more information about shared-memory communication, refer to “Where the Client Attaches to the Communications Portion” on page 14-11

Network Connections

You must use a network connection when the client 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.

OnLine supports the following types of network connections:

- TCP/IP using sockets
- TCP/IP using TLI
- IPX/SPX using TLI

The machine notes file that is part of your installation package tells which connection types are supported for your platform. (See “Useful On-Line Files” on page 9 of the Introduction.)

Connection Between Two Computers

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

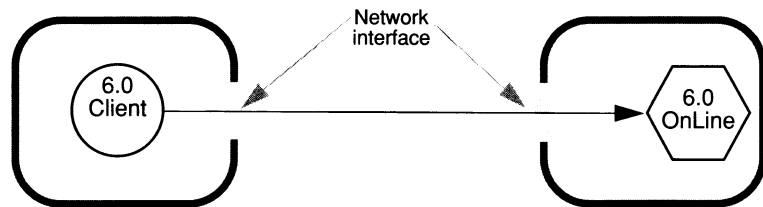


Figure 4-2 A network connection

Local Loopback 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 and the database server were on different computers. You cannot make a local loopback connection unless 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. It is as if the information passes from the network connection of the client, out of the computer, and then back in again through the network connection of the database server.

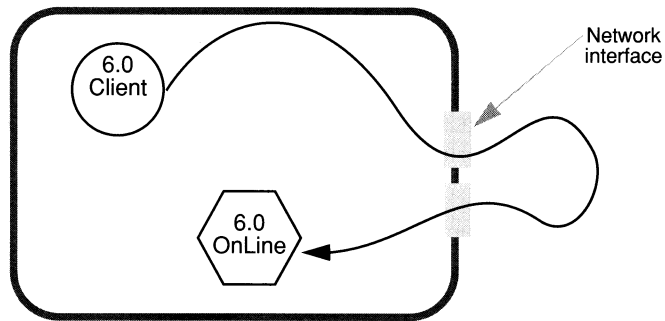


Figure 4-3 *Conceptual illustration of local loopback connection*

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 4-4 only in the type of connection between the client application and the database server.

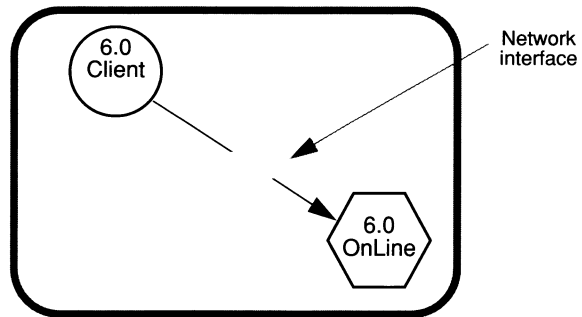


Figure 4-4 A local loopback connection

Connectivity Files

The *connectivity files* contain the information that enables a client application to communicate with a database server. 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 **OnLine** administrator manages only the `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 managed by the UNIX system (or network) administrator or by the end user.

Network Configuration Files

OnLine supports both TCP/IP and IPX/SPX communications protocols. You need different network configuration files for the different communication protocols.

Using the TCP/IP Communication Protocol

When you use the TCP/IP communication protocol, you use information from the network configuration files `/etc/hosts` and `/etc/services` to prepare the `sqlhosts` file. The network administrator maintains these files, so you need to work closely with the network administrator to make sure the information 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
- hostname
- host aliases (optional)

While the length of the hostname is not limited in the `/etc/hosts` file, it is limited to 64 characters in the `sqlhosts` file. Figure 4-8 on page 4-15 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 containing 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 file and must be identical on all computers running Informix client/server products. The aliases field is optional. For example, an `/etc/services` file might include the following entry for an **OnLine** database server:

online2	1526/tcp
---------	----------

This entry makes an **OnLine** database server known across the network as one of the services available for authorized users. Figure 4-8 on page 4-15 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 are not used and may not be up to date. You can view the contents of the NIS files by entering these commands at the system prompt: `yycat hosts` and `yycat services`.*

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

Using IPX/SPX Connections

When you use the IPX/SPX connections, the network configuration files vary from one vendor to another. Talk with the network administrator of your operating system to find out what files are required.

Network Security Files

Informix products follow standard UNIX security procedures, 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/password` and, if appropriate, `/etc/shadow`).

Users can explicitly specify the user ID and password that is used for connection to the remote computer by putting entries in the `.netrc` file in their home directory. The client application can specify a user ID and password in the `USER` clause of the `CONNECT` statement. If a user has specified an ID in the `~/.netrc` file and the client application has also specified an ID, the user ID and password specified by the client application takes precedence. For more information about the `CONNECT` statement, refer to the *Informix Guide to SQL: Syntax*.

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

The `/etc/hosts.equiv` and `~/.rhosts` files are optional files on the computer running the database server. They specify which remote hosts and user are trusted by that host. Trusted users are allowed to access the system 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 `/etc/hosts.equiv` file applies to the entire system. Individual users can maintain their own `.rhosts` file in their home directories.

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`.

The `~/.netrc` File

If the client does not provide a user ID and password with the `USER` clause of the `CONNECT` statement, the database server looks up a user ID and password in the `.netrc` file, if one is available.

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

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

The `$INFORMIXDIR/etc/sqlhosts` File

The `sqlhosts` file contains information that lets a client application find and connect to an Informix database server anywhere on the network. The `sqlhosts` file contains an entry (one line) for each type of connection to each database server on the network. Each entry in the `sqlhosts` file has the following four fields, which are covered in detail in the next sections:

- The `dbservername` field
- The `nettype` field
- The `hostname` field
- The `servicename` field

Figure 4-5 shows a sample **sqlhosts** file.

dbservername field	nettype field	hostname field	servicename field
menlo	onipcshm	valley	shm_file
menlo2	ontlitcp	valley	menlo_on
newyork	ontlitcp	hill	online2
pittsburgh	onsoctcp	canyon	online3

Figure 4-5 Sample sqlhosts file

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

```
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 **localinfo**, the client application would issue the following statement:

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

The **CONNECT** statement is fully documented in the *Informix Guide to SQL: Syntax*.

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

Editing the *sqlhosts* File

You can edit the **sqlhosts** file using any convenient text editor. For entries that refer to **OnLine** database servers, 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 4-14.
- 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 64 characters, and the **servicename** field is limited to 128 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 4-17. 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.

OnLine uses the dbservername as the index to obtain the connectivity information in the remaining fields when you initialize **OnLine** and when client applications connect to database servers.

The *nettype* Field

The **nettype** (network protocol type) field describes the type of connection that should be made between the client application and the database server.

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

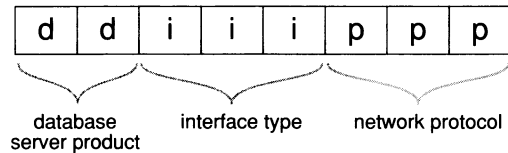


Figure 4-6 *Format of the nettype field*

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	OnLine (this is the more common form)
ol	OnLine
se	INFORMIX-SE
dr	INFORMIX-Gateway with DRDA

The Middle Three Letters of *nettype*

The middle three letters of **nettype** represent the interface that enables communications, as follows:

ipc	IPC (interprocess communication)
soc	sockets
tli	TLI (transport level interface)

Interprocess communications (IPC) are UNIX-based connections used only for communications between two processes running on the same computer. (For information about IPC connections, refer to “The Communications Portion of OnLine Shared Memory” on page 14-29.) The software that provides the interface between a program and the network protocol driver is the *network interface*. Informix supports two network interfaces: TLI and sockets.

The Final Three Letters of *nettype*

The final three letters of **nettype** represent the specific IPC mechanism or the network protocol, as follows:

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

IPC connections for the **OnLine** database server use shared memory. The rules or conventions for the behavior of networks are called *network protocols*. Informix supports two network protocols: TCP/IP and IPX/SPX. The IPX/SPX protocol is usually supported only on the TLI interface.

Figure 4-7 summarizes the **nettype** values for **OnLine**:

nettype	Description	Connection Type
onipcshm	OnLine using shared-memory communication (IPC)	(IPC)
ontlitcp	OnLine using TLI with TCP/IP protocol	(network)
onsoctcp	OnLine using sockets with TCP/IP protocol	(network)
ontlisp	OnLine using TLI with IPX/SPX protocol	(network)

Figure 4-7 Summary of *nettype* values for *OnLine*

The *hostname* Field

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

Shared-Memory Communication

When you use shared-memory communication, you can choose an arbitrary value for the **hostname** field. However, using the actual *hostname* makes your **sqlhosts** file easier to maintain.

Network Communication Using TCP/IP

When you use the TCP/IP connection protocol, the **hostname** field serves as a key into the **/etc/hosts** file, which gives the computer's network address. 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.

\$INFORMIXDIR/etc/sqlhosts file

dbservername	nettype	hostname	servicename
menlo2	ontlitcp	valley	menlo_on
newyork	ontlitcp	hill	online2
pittsburgh	onsoctcp	canyon	online3

/etc/hosts file

net address	hostname	host alias(es)
29.9.925.6	hill	sales
66.9.30.62	canyon	acctg
35.14.30.43	valley	

Figure 4-8 Relationship of *sqlhosts* file with the */etc/hosts* file when using TCP/IP

Network Communication Using IPX/SPX

When you use the IPX/SPX connection protocol, the **hostname** field of the **sqlhosts** file contains 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.

*Note: The display screens associated with NetWare installation utilities display the NetWare file server name in capital letters, for example: VALLEY. However, in the **sqlhosts** file, you can enter the name in either upper or lower case letters.*

The *servicename* Field

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

Shared-Memory Communication

When you use shared-memory communication, **OnLine** uses the **service-name** entry internally to look up the name of a file that contains shared-memory information. The **servicename** field for a shared-memory connection can be any value that is unique on the server computer.

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 4-9. 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 **servicename** you choose, as long as you agree on a name with the network administrator.

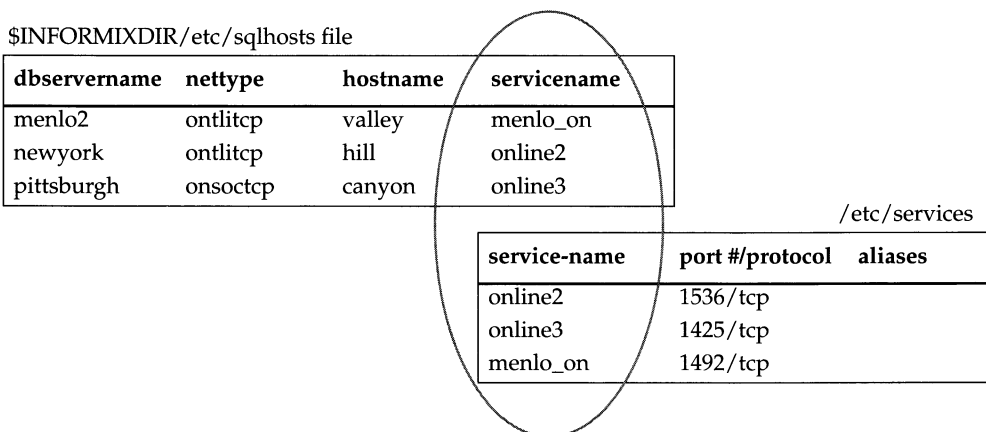


Figure 4-9 Relationship of *sqlhosts* file and *etc/services* file when using TCP/IP

Network Communication Using IPX/SPX

A *service* on the IPX/SPX network is simply a program that is prepared to do work for you, such as an **OnLine** database server. For an IPX/SPX connection, the value in the **servicename** field can be an arbitrary string, but it must be unique among the names of services available on the IPX/SPX network. It is convenient to use the dbservername in the **servicename** field. When you use **INFORMIX-OnLine for NetWare 4.1**, the **servicename** *must* be the same as the dbservername.

ONCONFIG Parameters for Connectivity

When you initialize an **OnLine** database server, the initialization procedure uses parameter values from the ONCONFIG configuration file. (For a general discussion of **OnLine** 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.

The NETTYPE parameter is not a required parameter. It lets you adjust the number and type of virtual processors used by the database server for communication. After your **OnLine** database server has been running for some time, you might want to 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 12-24.

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 dbservername in the following places:

- In the INFORMIXSERVER environment variable
- In SQL statements such as CONNECT, DATABASE, CREATE TABLE, and ALTER TABLE. For example:

```
CONNECT TO '@nyc_research'
```

- In the DBPATH environment variable

The DBSERVERALIASES Configuration Parameter

The DBSERVERALIASES parameter lets you assign multiple dbservernames to the same **OnLine** database server. Figure 4-10 shows entries in an ONCONFIG configuration file that assign three dbservernames to the same **OnLine** database server.

```
DBSERVERNAME    sockets_online
DBSERVERALIASES ipx_online,shm_online
```

Figure 4-10 Example of DBSERVERNAME and DBSERVERALIASES parameters

The **sqlhosts** file associated with the dbservernames from Figure 4-10 could include the entries shown in Figure 4-11. Since each dbservername has a corresponding entry in the **sqlhosts** file, you can associate multiple connection types with one database server.

```
shm_online      onipcshm      my_host        my_shm
sockets_online  onsoctcp      my_host        port1
ipx_online      ontlispdx     nw_file_server ipx_online
```

Figure 4-11 Three entries in the sqlhosts file for one OnLine database server

Using the **sqlhosts** file shown in Figure 4-11, a client application uses the following statement to connect to the database server using shared-memory communication:

```
CONNECT TO '@shm_online'
```

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

```
CONNECT TO '@sockets_online'
```

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 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 **OnLine**. Users of client applications that connect to **OnLine** using network connections might need to set these variables.

If the client application explicitly attaches to shared-memory segments, you may need to set `INFORMIXSHMBASE` (shared-memory base). Refer to “Where the Client Attaches to the Communications Portion” on page 14-11.

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 6.0 database servers
- Using the 6.0 relay module
- Using 5.0 **INFORMIX-STAR** or 5.0 **INFORMIX-NET**
- Using a 6.0 client with a 5.0 server

For more information about client/server connections, refer to “How a Client Attaches to the Communications Portion” on page 14-10.

Note: In the following examples you can assume that the network-configuration files `letclhosts` and `letclservices` have been correctly prepared, even if they are not explicitly mentioned.

Using a Shared-Memory Connection

Figure 4-12 shows a shared-memory connection on the computer named `river`.

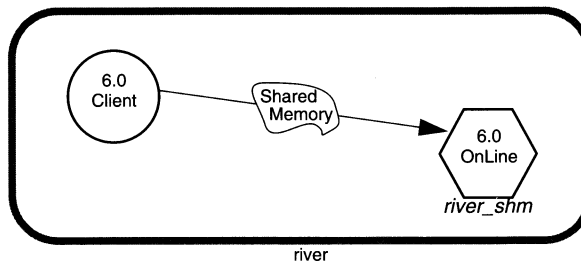


Figure 4-12 A shared-memory connection

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

```
DBSERVERNAME river_shm
```

The following table shows a correct entry for the `sqlhosts` file:

<code>dbservername</code>	<code>nettype</code>	<code>hostname</code>	<code>servicename</code>
<code>river_shm</code>	<code>onipcshm</code>	<code>river</code>	<code>rivershm</code>

The client application connects to this database server using the 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 4-13 shows a local loopback connection. The name of the host computer is **river**.

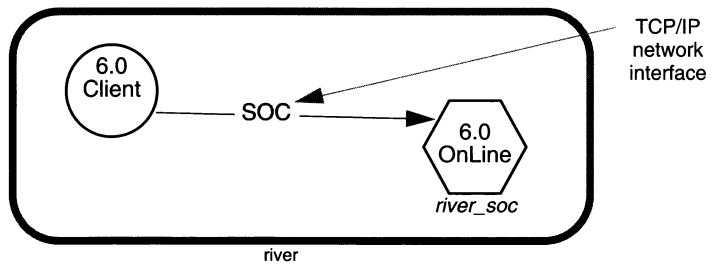


Figure 4-13 A local loopback configuration

The network connection in Figure 4-13 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 **ontlitcp** instead of **onsoctcp**.

The ONCONFIG file includes the following line:

```
DBSERVERNAME river_soc
```

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

Using a Network Connection

Figure 4-14 shows a configuration where the client application resides on host **river** and the **OnLine** database server resides on host **valley**.

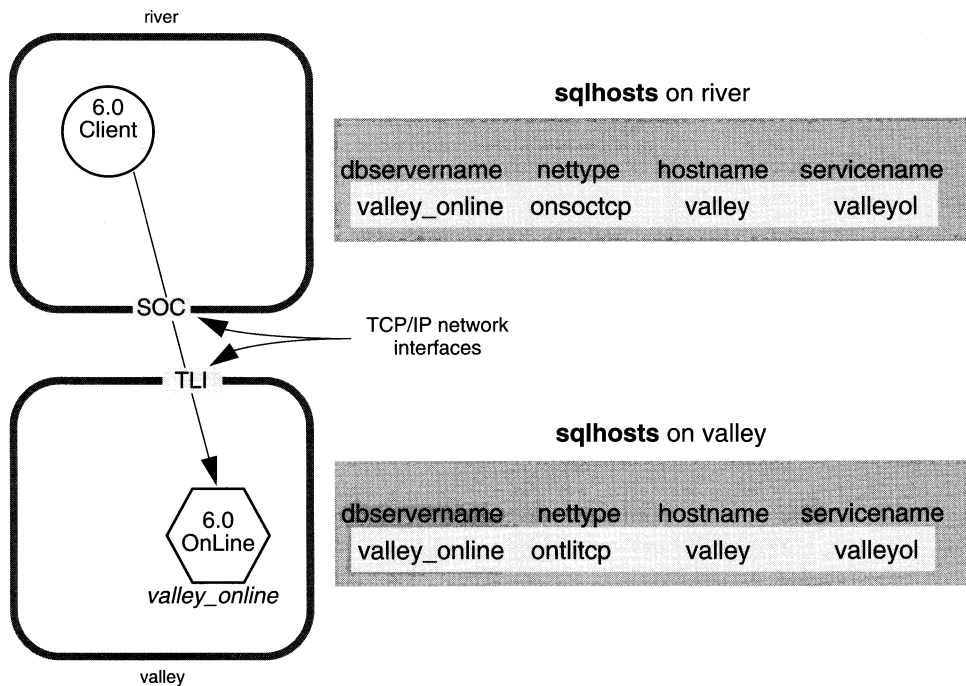


Figure 4-14 A network configuration

Note that an entry for the `valley_online` 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 interface, while the host **valley** uses TLI for its network interface. The **nettype** field must reflect the type of network interface used by the computer on which the `sqlhosts` file resides. In this example, the **nettype** field for the `valley_online` database server on host **river** is `onsoctcp`, while the **nettype** field for the `valley_online` database server on host **valley** is `ontlitcp`.

The *sqlhosts* File Entry for an IPX/SPX Network

IPX/SPX usually uses a TLI interface. If the configuration in Figure 4-14 on page 4-22 uses an IPX/SPX network instead of a TCP/IP network, the entry in the *sqlhosts* file on both computers is as follows:

dbservername	nettype	hostname	servicename
valley_on	ontlisp	valley_nw	valley_on

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

An **OnLine** database server can provide more than one type of connection. Figure 4-15 on page 4-24 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 an **OnLine** database server supports 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 4-15, the database server has two dbservernames: *river_net* and *river_shm*. The ONCONFIG configuration file includes the following entries:

DBSERVERNAME	river_net
DBSERVERALIASES	river_shm

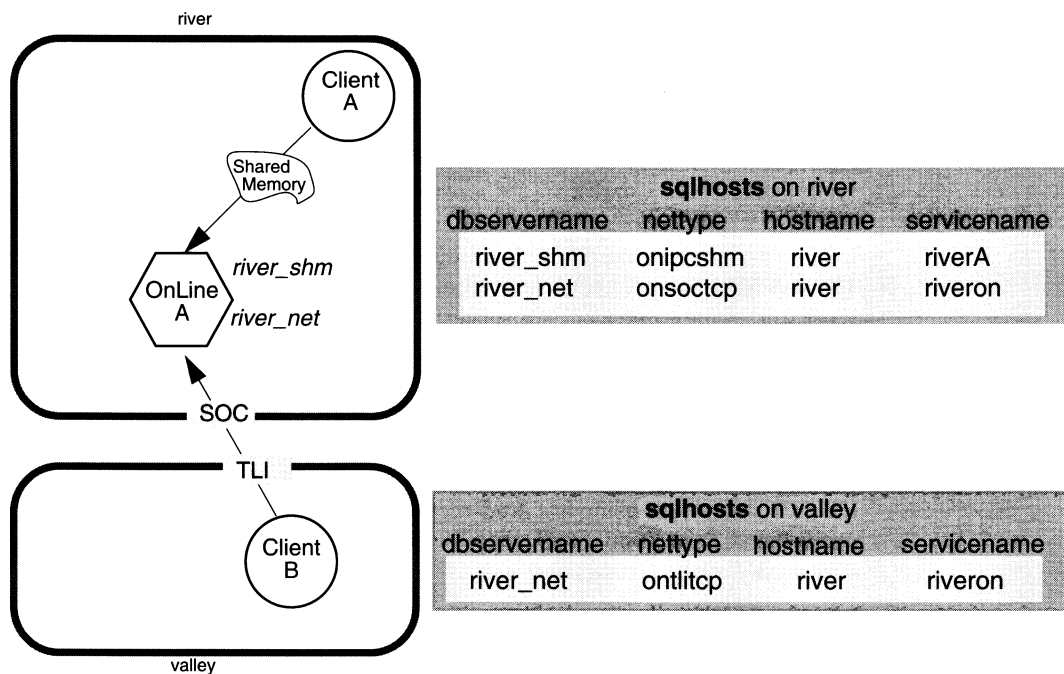


Figure 4-15 A configuration using multiple connection types

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 6.0 OnLine Database Servers

Figure 4-16 shows a configuration with two **OnLine** database servers on host **river**. When more than one **OnLine** database server is active on one computer, it is known as *multiple residency*. (See Chapter 5, "What Is Multiple Residency?" for information about multiple residency.)

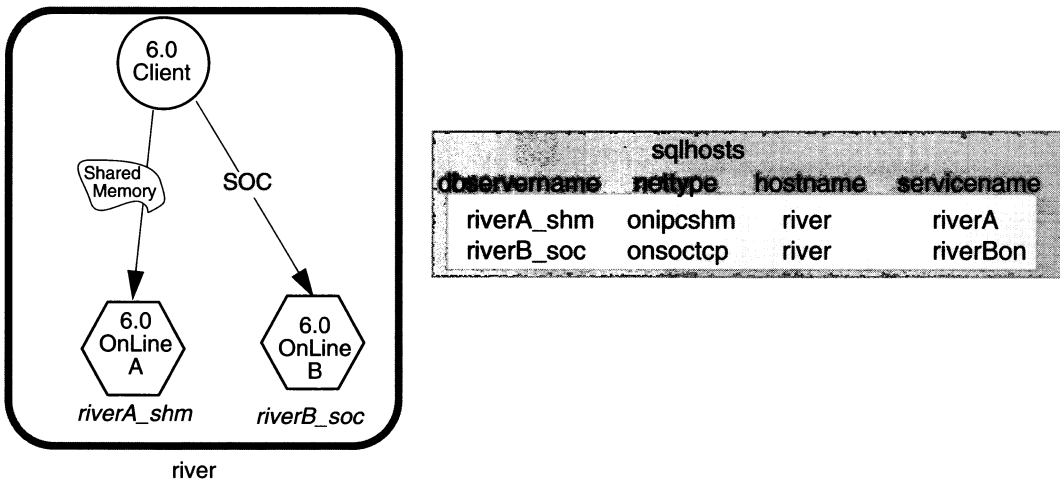


Figure 4-16 Multiple OnLine database servers

For the configuration in Figure 4-16, you must prepare two ONCONFIG configuration files, one for **OnLine A** and the other for **OnLine B**. The `sqlhosts` file includes the connectivity information for both **OnLine** database servers.

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

```
DBSERVERNAME    riverA_shm
```

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

```
DBSERVERNAME    riverB_soc
```

Using the 6.0 Relay Module

Every Version 6.0 Informix database server includes a *relay module* that lets Version 5.0 or Version 4.1 client applications connect to a local 6.0 database server. The 6.0 relay module is used when a pre-6.0 client application connects to a local 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** or the 6.0 relay module. The relay module serves a very important function—allowing connections between Informix products from different release levels—but it is designed to be as invisible as possible.

Figure 4-17 shows an example of a 5.0 client application connected to a 6.0 **OnLine** database server using the 6.0 relay module.

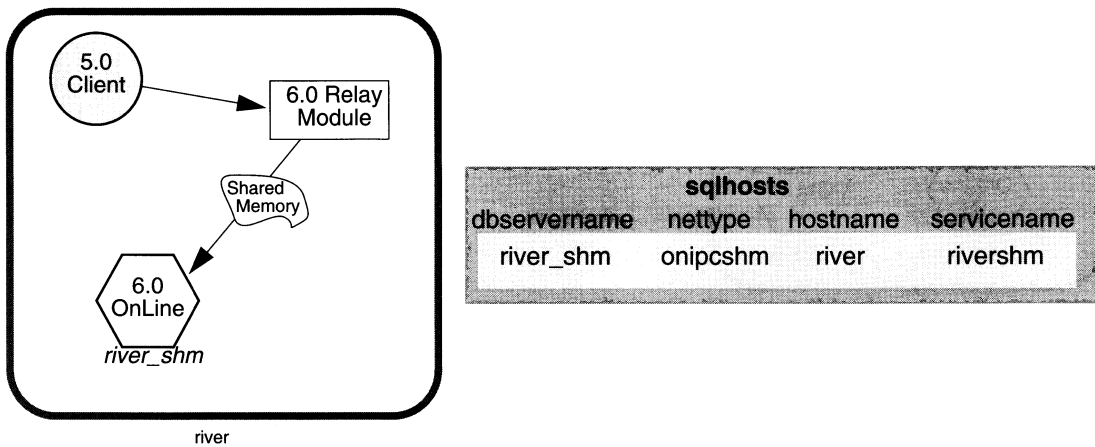


Figure 4-17 A configuration with a 5.0 client application and a 6.0 database server

To prepare this configuration, follow this process:

1. Verify that the **OnLine** database server works correctly with a 6.0 client application that uses a shared-memory connection. (In other words, pre-

pare your configuration as you would for the configuration shown in Figure 4-12 on page 4-20.)

Note that 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. If 5.0 Informix products are installed in the same directory as 6.0 products, set the `SQLEXEC` environment variable to the pathname of the 6.0 relay module. (The relay module is stored as `$INFORMIXDIR/lib/sqlrm` as part of the installation process.) For example:

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

3. If the 5.0 Informix products are in a different directory from the 6.0 products, take the following actions:
 - o Change the `INFORMIXDIR` environment variable to point to the directory where the 5.0 products are installed.
 - o Modify the `PATH` environment variable to include `$INFORMIXDIR/bin`.
 - o Set the `SQLEXEC` environment variable to the complete pathname of the relay module. You cannot use the variable `$INFORMIXDIR` to set the `SQLEXEC` environment variable, because the `INFORMIXDIR` environment variable now points to the directory of the 5.0 products, instead of to the directory where the 6.0 products are stored. You must use the exact pathname, such as:

```
setenv SQLEXEC /usr/version6/informix/lib/sqlrm
```

4. Remove extra environment variables

If Version 5.0 Informix products are in use, the user's environment might include two environment variables that were required for Version 5.0 database servers: `SQLRM` and `SQLRMDIR`. The user must unset these variables before the client application can use the 6.0 **OnLine** database server. For example:

```
unsetenv SQLRM
unsetenv SQLRMDIR
```

The DBNETTYPE environment variable, used by the 5.0 database servers, is not needed for the 6.0 OnLine database server. You can unset the DBNETTYPE environment variable if you wish, but it does not affect Version 6.0 products in any way.

A Relay Module Configuration with Three Database Servers

Figure 4-18 shows an expanded version of the configuration in Figure 4-19. This configuration has three possible connections (called A, B, and C) between a 5.0 client application and 6.0 OnLine database servers. The client application can use any of the connections, but only one connection can be active at a time.

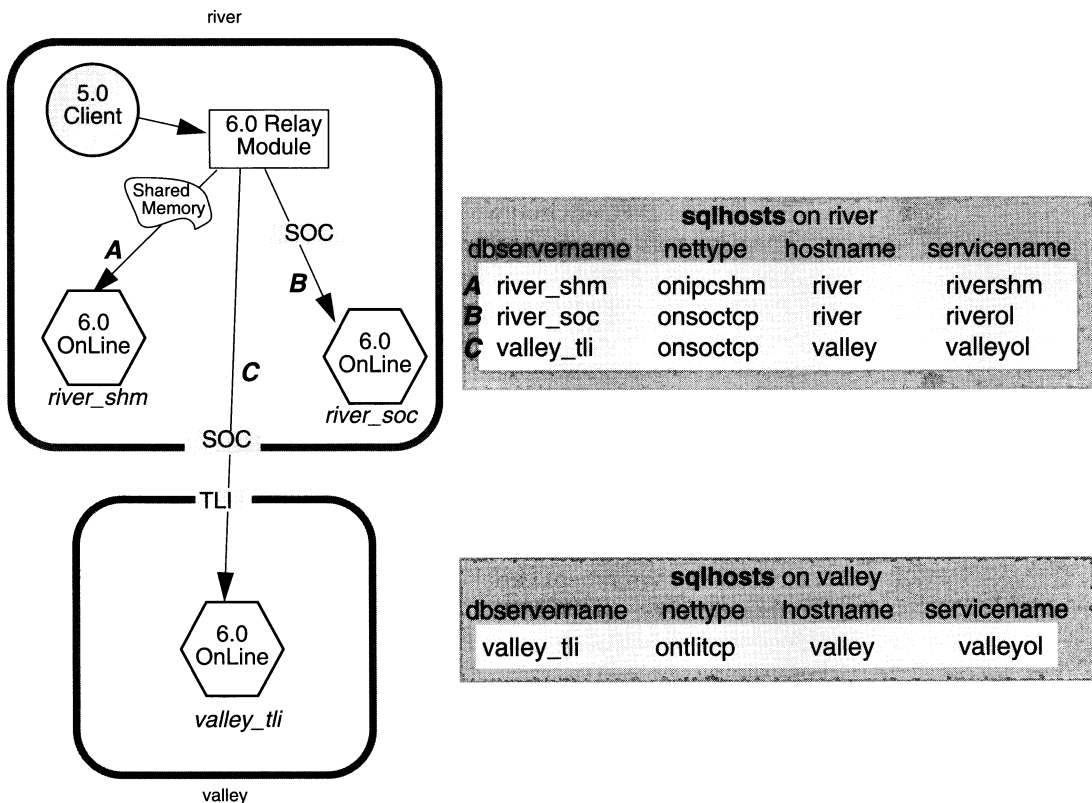


Figure 4-18 A configuration with a 6.0 relay module and three OnLine database servers

As with Figure 4-19, you should verify that all three connections work correctly with a 6.0 client application and then modify the environment variables.

Using 5.0 INFORMIX-STAR or 5.0 INFORMIX-NET

Figure 4-19 illustrates a 5.0 client application connecting to 6.0 OnLine using 5.0 OnLine with the INFORMIX-STAR client/server product.

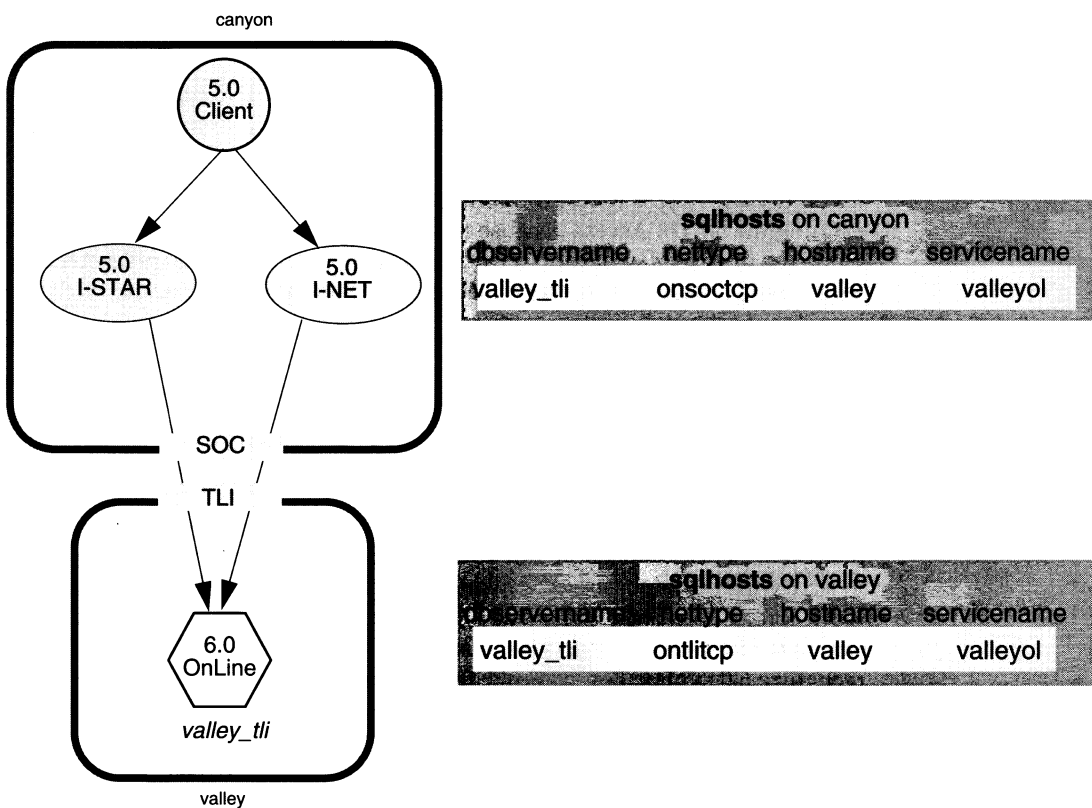


Figure 4-19 Example of a Version 5.0 client application with a Version 6.0 OnLine database server

To prepare to use the configuration in Figure 4-19, follow these steps:

1. On the server host (**valley**), start the 6.0 OnLine database server and verify that a 6.0 client application can connect to it using a local loopback

connection. (Refer to “Using a Local Loopback Connection” on page 4-21.)

2. Write down (for use in Step 4) the entry that is in the **sqlhosts** file.
3. On the client host (**canyon**), initialize the 5.0 **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 the **INFORMIX-STAR** or 5.0 **INFORMIX-NET** to contain an entry for the 6.0 **OnLine** database server.

Notice that the **sqlhosts** files for the configuration in Figure 4-19 are the same as they would be if the client were a 6.0 client. (Refer to “Using a Network Connection” on page 4-21.)

If you use **DB-Access** to test the configuration, you can verify that you are running the 5.0 **DB-Access** because the first display has only four choices, while the first display of the 6.0 **DB-Access** has six choices. When **DB-Access** asks for the database, enter the database name and the server name, such as:

```
my_database_name@valley_tli
```

Using a 6.0 Client Application with a 5.0 Database Server

You can connect a 6.0 client application to 5.0 **OnLine** with **INFORMIX-STAR** using a network connection, but you cannot connect a 6.0 client application to a 5.0 **OnLine** database server using shared memory. You must use a network connection. The 6.0 client application cannot use any syntax that is specific to 6.0 Informix products because the 5.0 database server does not recognize 6.0 syntax. For example, the 6.0 client application cannot use the **CONNECT** statement with a 5.0 database server because the **CONNECT** statement is specific to Version 6.0.

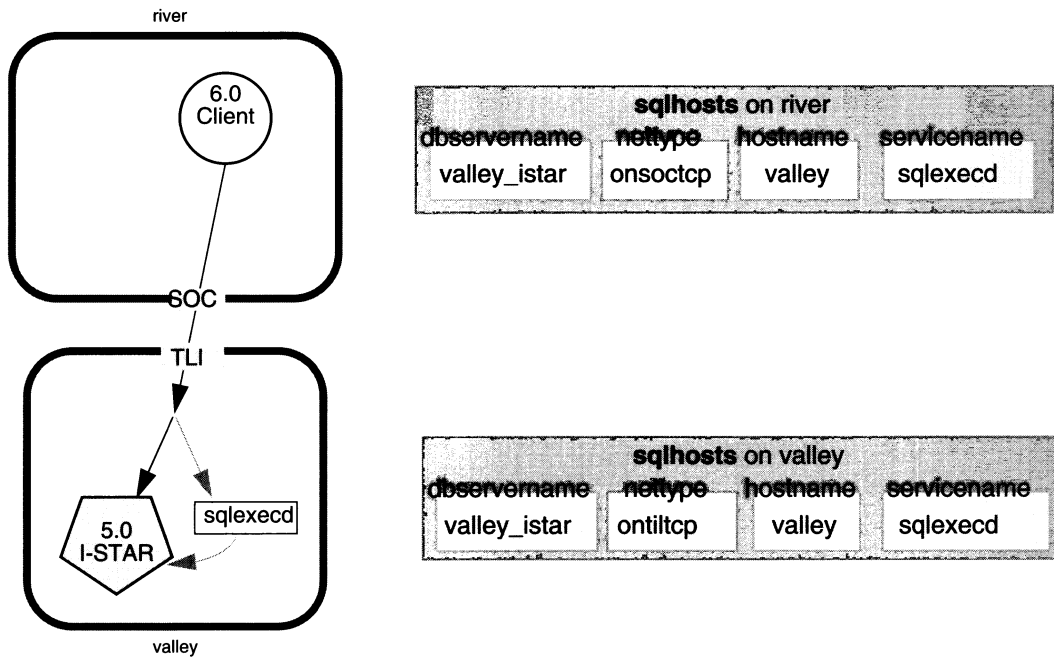


Figure 4-20 Example connecting a 6.0 client to a 5.0 database server

To prepare to use the configuration in Figure 4-20, follow these steps:

1. On the server host (**valley**), start 5.0 OnLine with **INFORMIX-STAR** and confirm that it is active by connecting a 5.0 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 5.0 OnLine with **INFORMIX-STAR**.
3. On the client host (**river**), set the **INFORMIXDIR** environment variable to the directory where the 6.0 Informix products are installed.
4. Modify the **PATH** environment variable to include **\$INFORMIXDIR/bin**.
5. Update the **sqlhosts** file to contain an entry for 5.0 OnLine with **INFORMIX-STAR**.
6. Set the **INFORMIXSERVER** environment variable to the **dbservername** of the 5.0 OnLine with **I-STAR**.

What Is Multiple Residency?

Chapter Overview 3

Benefits of Multiple Residency 3

How Multiple Residency Works 4

 The Role of the ONCONFIG Environment Variable 4

 The Role of the SERVERNUM Configuration
 Parameter 4

Chapter Overview

You can use more than one **INFORMIX-OnLine Dynamic Server** database server in the following two ways:

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

When multiple **OnLine** 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 **OnLine** database server environments on the same computer allows you to:

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

When you use multiple residency, each **OnLine** database server has its own configuration file. Thus, you can create a configuration file for each database server that meets its special requirements for archiving, 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 reasons of security or to accommodate more frequent archiving than is required for the majority of the databases.

If you are developing an application for use on a network, you can use local loopback (see “Using a Local Loopback Connection” on page 4-21) 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.

How Multiple Residency Works

Multiple residency is possible because the UNIX operating system can maintain separate areas in UNIX shared memory for each instance of **OnLine**. Each instance of **OnLine** 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 **OnLine**.

The Role of the **ONCONFIG** Environment Variable

Each instance of **OnLine** 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 **OnLine** 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

You maintain separation between the instances of **OnLine** database servers by maintaining multiple configuration files, each with a unique **SERVERNUM** value. When an **OnLine** database server is initialized, **OnLine** reads the **ONCONFIG** environment variable for the name of its configuration file. Next, **OnLine** reads its configuration file to obtain the value of its **SERVERNUM** parameter. **OnLine** 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 online_one	DBSERVERNAME online_two
SERVERNUM 1	SERVERNUM 2
ROOTPATH /dev/area1	ROOTPATH /dev/area2
...	...

Figure 5-1 illustrates 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.

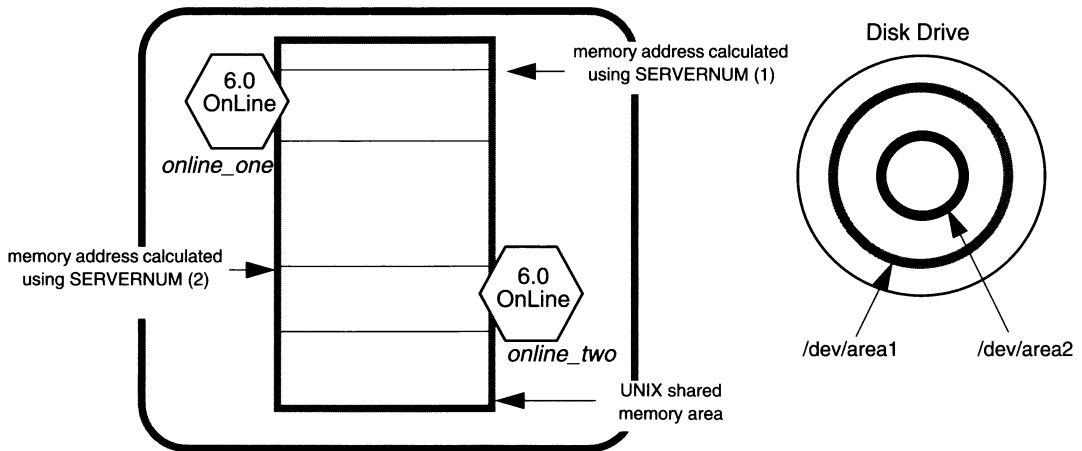


Figure 5-1 Illustration of separate memory and storage in multiple residency

Using Multiple Residency

Chapter Overview 3

Planning for Multiple Residency 3

Preparing for Multiple Residency 4

 Prepare a Configuration File 4

 Set Your ONCONFIG Environment Variable 4

 Edit the New Configuration File 5

 Add Connection Information 6

 Update the `$INFORMIXDIR/etc/sqlhosts` File 6

 Initialize Disk Space 6

 Prepare Archive and Backup Environment 7

 Update the Operating System Boot File 8

 Check Users' INFORMIXSERVER Environment
 Variables 8

Chapter Overview

This chapter describes how to use multiple **INFORMIX-OnLine Dynamic Server** database servers on the same computer. It describes the following topics:

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

Before you perform this procedure, you should already have installed one **OnLine** database server as described in Chapter 3, “Installing and Configuring OnLine.”

Planning for Multiple Residency

Each **OnLine** database server must have its own unique storage space. You cannot use the same disk space for more than one instance of **OnLine**. When you prepare an additional **OnLine** database server, you need to repeat some of the planning you did for installing the first **OnLine** 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 be?
- Can you dedicate a tape drive to this database server for its logical logs?
- What kind of archiving will you do?

Preparing for Multiple Residency

*Note: Do not try to install another copy of the **OnLine** binaries. All instances of the same version of **OnLine** on one host computer share the same executables.*

Here is a summary of the steps for creating another **OnLine** database server on a computer that already has one **OnLine** 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 dserver-name(s) of the new database server.
6. Initialize disk space for the new database server.
7. Prepare archive 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 **OnLine** must have its own ONCONFIG configuration file. You prepare an ONCONFIG file for the new instance of **OnLine** by copying a configuration file that already exists and modifying it appropriately. You can copy a configuration that you have already prepared or 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 the SERVERNUM in the file (using a text editor) before entering 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:

- ROOTPATH and/or ROOTOFFSET

The ROOTPATH and ROOTOFFSET parameters together specify the location of the root dbspace for this **OnLine** database server. The root dbspace location must be unique for every **OnLine** 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. Refer to “ROOTPATH” on page 35-36 and “ROOTOFFSET” on page 35-35 for more information.

Note: 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.

- SERVERNUM

The SERVERNUM parameter specifies an integer (between 0 and 255) associated with this **OnLine** configuration. Each instance of **OnLine** on the same host computer must have a unique SERVERNUM. Refer to “The Role of the SERVERNUM Configuration Parameter” on page 5-4 for more information.

- DBSERVERNAME

The DBSERVERNAME parameter specifies the dbservername of this **OnLine** database server. Informix suggests that you choose a name that gives information about the database server, such as **ondev37** or **hostnamedev37**. Refer to “The DBSERVERNAME Configuration Parameter” on page 4-17 for more information.

- MSGPATH

The MSGPATH parameter specifies the UNIX pathname of the message file for this **OnLine** database server. You should specify a unique pathname for the message file because **OnLine** messages do not include the dbservername. If multiple **OnLine** database servers use the same MSGPATH you will not be able to identify the messages from separate **OnLine** 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 **OnLine**.

You might need to set these parameters:

- **MIRRORPATH** and/or **MIRROROFFSET**

If the root dbspace is mirrored, the location of the root dbspace mirror must be unique. Refer to “Steps Required for Mirroring Data” on page 24-3 for information about setting **MIRRORPATH**.

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 **OnLine**. 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 **OnLine**, the administrators on those computers must update their **sqlhosts** files. Chapter 4, “Configuring Connectivity,” discusses the preparation of the **sqlhosts** file.

If you plan to use TCP/IP network connections with this instance of **OnLine**, 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 “Using the TCP/IP Communication Protocol” on page 4-8 and “Using IPX/SPX Connections” on page 4-9.

Initialize Disk Space

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



As you create new blobspaces and/or dbspaces for this **OnLine** database server, be sure you assign each chunk to a unique location on the device. **OnLine** does not allow you to assign more than one chunk to the same loca-

tion within a single **OnLine** environment, but it remains your responsibility as administrator to make sure that chunks belonging to *different OnLine* database servers do not overwrite each other.

Prepare Archive and Backup Environment

This section gives a very brief discussion of the effects of multiple residency on archiving and backups. For more information, refer to the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*.

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

Each **OnLine** 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 is serving the different **OnLine** instances before you kill them. The *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* explains how to associate the process id of an **oncatlgr** process with an **OnLine** instance.

Every **OnLine** instance running out of the same **\$INFORMIXDIR** uses the same **config.arc** file. This is different from ONCONFIG files, where an environment variable points to a different file for each instance. The parameters in **config.arc** are of such a general nature that sharing them between instances is usually not a problem. If you want to use different archiving defaults for different **OnLine** instances, the users might need to set their ARC_DEFAULT environment variables to point to a different defaults file.

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 3-28) so that each of your **OnLine** instances starts whenever the computer is rebooted (for example, after a power failure).

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

Check Users’ INFORMIXSERVER Environment Variables

If this new instance of **OnLine** 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.

The image features a dark, textured background with a vertical light streak running down the center. The texture is grainy and appears to be a scan of a physical surface. The light streak is brighter in the middle and fades towards the top and bottom.

Modes and Initialization



What Are OnLine Operating Modes?

Chapter Overview 3

Off-Line Mode 3

Quiescent Mode 3

On-Line Mode 4

Read-Only Mode 4

Recovery Mode 4

Shutdown Mode 4

Chapter Overview

This chapter explains the operating modes of **INFORMIX-OnLine Dynamic Server**.

You can determine the current **OnLine** mode by executing **onstat**. The mode is displayed in the header. The mode also appears in the status line displayed in ON-Monitor. **OnLine** 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 **OnLine** modes, see Chapter 8, “Managing Modes.”

Off-Line Mode

When **OnLine** is in off-line mode, it is not running.

Quiescent Mode

Administrative procedures that require a pause in database activity are performed when **OnLine** 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 **OnLine** is in on-line mode, users can connect with the database server and perform all database activities. The **OnLine** administrator (user **informix** or user **root**) can use the command-line utilities to change many **OnLine** ONCONFIG parameter values while **OnLine** 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 **OnLine** is moving from off-line to quiescent mode. *Fast recovery* is performed when **OnLine** is in recovery mode.

(It is possible for a mirrored chunk to be in recovery *state*, but this is not the same as **OnLine** recovery *mode*.)

Shutdown Mode

Shutdown mode is transitory. It occurs when **OnLine** 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

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

Chapter Overview

This chapter contains instructions on changing **INFORMIX-OnLine Dynamic 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 OnLine Operating Modes?”

Users Permitted to Change Modes

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

From Off-Line to Quiescent

When **OnLine** changes from off-line mode to quiescent mode, **OnLine** initializes shared memory.

When **OnLine** is in quiescent mode, no sessions can gain access to **OnLine**. 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 **OnLine** 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 **OnLine** from off-line mode to quiescent mode.

To verify that **OnLine** 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 **OnLine** from off-line mode to on-line mode, **OnLine** initializes shared memory. You cannot go directly from off-line mode to on-line mode using ON-Monitor.

When **OnLine** is in on-line mode, it is accessible to all **OnLine** sessions.

How to Perform This Change Using *oninit*

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

To verify that **OnLine** 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 **OnLine** from quiescent mode to on-line mode, all sessions gain access.

If you have already taken **OnLine** from on-line mode to quiescent mode and you are now returning **OnLine** 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 **OnLine** 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 **OnLine** from quiescent mode to on-line mode.

To verify that **OnLine** 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 **OnLine** gracefully from on-line mode to quiescent mode to restrict access to **OnLine** without interrupting current processing.

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

How to Perform This Change Using ON-Monitor

To take **OnLine** 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 **OnLine** gracefully from on-line mode to quiescent mode.

To verify that **OnLine** 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 **OnLine** immediately from on-line mode to quiescent mode to restrict access to **OnLine** as soon as possible. Work in progress can be lost.

A prompt asks for confirmation of the immediate shutdown. If you confirm, **OnLine** 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, **OnLine** terminates this session.

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

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

How to Perform This Change Using ON-Monitor

To take **OnLine** 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 **OnLine** immediately from on-line mode to quiescent mode.

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

A prompt asks for confirmation to go off-line. If you confirm, **OnLine** 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 automatically comply within 10 seconds, **OnLine** terminates this session.

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

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

How to Perform This Change Using **ON-Monitor**

To take **OnLine** 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 **OnLine** off-line immediately.

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

What Is Initialization?

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

Chapter Overview

INFORMIX-OnLine Dynamic Server initialization 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 shared memory, as follows: **OnLine** 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, **OnLine** automatically initializes shared memory as part of the process.

When you initialize **OnLine** disk space, you overwrite whatever is on that disk space. If you reinitialize disk space for an existing **OnLine** database server, all of the data in the earlier **OnLine** 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 use **informix** or **root** to initialize **OnLine**. **OnLine** must be in off-line mode when you begin initialization. (See Chapter 7, “What Are OnLine Operating Modes?”)

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

- The **oninit** utility (See “oninit: Initialize OnLine” on page 37-16.)
- The ON-Monitor utility (See “Using ON-Monitor” on page 34-3.)

The options you include in the **oninit** command or the options 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 with a newly initialized disk those activities are not relevant.

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.

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

Figure 9-1 Initialization steps

Process Configuration File

OnLine 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 **OnLine** to off-line mode and then reinitialize.

During initialization, **OnLine** looks for configuration values in the following three files, in order:

1. If the ONCONFIG environment variable is set, **OnLine** reads values from the file specified by `$INFORMIXDIR/etc/$ONCONFIG`. If the ONCONFIG

environment variable is set, but **OnLine** cannot access the specified file, **OnLine** returns an error message.

2. If the ONCONFIG (or TBCONFIG) environment variable is not set, **OnLine** reads the configuration values from the file `$INFORMIXDIR/etc/onconfig`.
3. If **OnLine** cannot find the `onconfig` file, it reads the configuration values from `$INFORMIXDIR/etc/onconfig.std`.

Informix recommends that you *always* set the ONCONFIG environment variable before initializing **OnLine**. The default configuration files are intended as templates and not as functional configurations. (See “Prepare the ONCONFIG Configuration File” on page 3-13.)

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. (See “PAGE_CONFIG” on page 40-8.) Where differences exist, **OnLine** uses the values from the current ONCONFIG configuration file for initialization.

Create Shared-Memory Segments

Next, **OnLine** uses the configuration values to calculate the required size of **OnLine** resident shared memory. In addition, **OnLine** computes additional configuration requirements from internal values. Space requirements for overhead are calculated and stored.

OnLine creates shared memory by acquiring the shared-memory space from the operating system for three different types of memory segments:

- The resident segment, used for data buffers, tablespaces, and so on
- Message segments, used for communication
- Virtual segments, used to track specific tasks for individual users

Next, **OnLine** attaches the shared-memory segments to its virtual address space and initializes shared-memory structures. (See “The Virtual Portion of OnLine Shared Memory” on page 14-25.)

After initialization is complete and **OnLine** is running, it can create additional shared-memory segments as needed. **OnLine** creates segments in increments of the page size.

Initialize Shared-Memory Structures

After attaching to shared memory, **OnLine** clears the shared-memory space of uninitialized data. Next **OnLine** lays out the shared-memory header information and initializes data in the shared-memory structures. For example, **OnLine** 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. (See “onstat: Monitor OnLine Operation” on page 37-46.)

After **OnLine** 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. **OnLine** 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

Note: This procedure is done only during disk-space initialization.

After shared-memory structures are initialized, **OnLine** begins initializing the disk. **OnLine** initializes all the reserved pages that it maintains in the root dbspace on disk and writes PAGE_PZERO control information to the disk. (See “Reserved Pages” on page 40-6.)

Start All Required Virtual Processors

OnLine 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. (See “What Is a Virtual Processor?” on page 12-4.)

Make Necessary Conversions

OnLine checks its internal files. If the files are from an earlier version of **OnLine**, it updates these files to 6.0 format. For information about database conversion, refer to the *INFORMIX-OnLine Dynamic Server Migration Guide*.

Initiate Fast Recovery

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

OnLine checks if fast recovery is needed. If fast recovery is required, **OnLine** initiates fast recovery. (See “What Is Fast Recovery?” on page 22-3.)

Initiate a Checkpoint

After fast recovery executes, **OnLine** initiates a checkpoint. As part of the checkpoint procedure **OnLine** writes a checkpoint complete message in the **OnLine** message log. (See “OnLine Checkpoints” on page 14-47.)

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

Document Configuration Changes

Note: This task is not done during disk-space initialization.

OnLine 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, **OnLine** notes both values (old and new) in a message to the **OnLine** message log.

Create the *oncfg_servername.servernum* File

OnLine creates the *oncfg_servername.servernum* file and updates it every time you add or delete a dbspace, a blobspace, a logical-log file, or a chunk. You do *not* need to manipulate this file in any way, but you can see it listed in your \$INFORMIXDIR/etc directory. **OnLine** uses this file during a full-system restore. (See “oncfg_servername.servernum” on page 42-8.)

Drop Temporary Tblspaces

Note: This task is not done during disk-space initialization.

OnLine searches through all dbspaces searching for temporary tblspaces. (If you initialize **OnLine** using **oninit -p**, **OnLine** skips this step.) These temporary tblspaces (if any) are tblspaces left by user processes that died prematurely and were unable to perform proper cleanup. **OnLine** deletes any temporary tblspaces and reclaims the disk space. (See “What Is a Temporary Table?” on page 10-23.)

Set Forced Residency, If Specified

If the value of the `RESIDENT` configuration parameter is one, **OnLine** 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 **OnLine** sends an error message to the message log. (See “`RESIDENT`” on page 35-35.)

Return Control to User

After the previous steps are complete, **OnLine** writes an “initialization complete” message in the **OnLine** message log. (See “`MSGPATH`” on page 35-27.)

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 **OnLine** message log.

Prepare SMI Tables

Even though **OnLine** has returned control to the user, it has not finished its work. **OnLine** now checks the *system monitoring interface* (SMI) tables. (See Chapter 36, “The sysmaster Database.”) If the SMI tables are not current, **OnLine** updates the tables. If the SMI tables are not present, as is the case when disk is initialized, **OnLine** creates the tables. After **OnLine** builds the SMI tables, it puts the message `sysmaster database built successfully` into the **OnLine** message log file.

If you shut down **OnLine** before **OnLine** finishes building the SMI tables, the process of building the tables aborts. This does not damage **OnLine**. **OnLine** simply builds the SMI tables the next time you bring **OnLine** 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-Archive for backups or archives.

After Initialization

After the SMI tables have been created, **OnLine** is ready for use. **OnLine** runs until you stop it using `onmode` or ON-Monitor or the system crashes. Informix recommends that you *do not* try to stop **OnLine** by killing a virtual processor or an oninit process. (See “Starting and Stopping Virtual Processors” on page 13-6.)



Disk, Memory, and Process Management



Where Is Data Stored?

Chapter Overview	3
Overview of Data Storage	3
What Are the Physical Units of Storage?	4
What Is a Chunk?	4
Should You Allocate Chunks as Cooked Files or Raw Disk Space?	5
What Is an Offset?	8
What Is a Page?	9
What Is A Blobpage?	10
How Big Should a Blobpage Be?	11
What Is an Extent?	11
What Are the Logical Units of Storage?	14
What Is a Dbspace?	15
How Can You Control Where Data Is Stored?	15
What Is the Root Dbspace?	17
What Is a Temporary Dbspace?	18
What Are the Advantages of Using Temporary Dbspaces?	18
What Is a Blobspace?	19
What Is a Database?	20
What Is a Table?	21
What Is a Tblspace?	25
What Is Extent Interleaving?	26

How Much Disk Space Do You Need to Store Your Data?	27
Calculate the Size of the Root Dbspace	27
Physical and Logical Logs	28
Temporary Tables	28
Data	28
ON-Archive Catalog Data	29
Control Information (Reserved Pages)	29
Complete the Root Dbspace Calculation	29
Estimate Space Required by Databases Including Overhead and Growth	30
Disk-Layout Guidelines	30
Strive to Associate Partitions with Chunks	31
Consider Mirroring	31
Isolate High-Use Tables	31
Group Your Tables with Archive and Restore in Mind	32
Spread a Single Table Across Multiple Disk Devices to Reduce Contention	33
Place High-Use Tables on Middle Partition of Disk	33
Spread Your Temporary Storage Space Across Multiple Disks	34
Optimize Table Extent Sizes	34
Move the Logical and Physical Logs from the Root Dbspace	35
Take into Account Archive and Restore Performance	36
Cluster Catalogs with the Data They Track	36
Reconsider Separating the Physical and Logical Logs	36
Sample Disk Layouts	37
Sample Layout When Performance Is Highest Priority	37
Sample Layout When Availability Is Highest Priority	40
Sample Layout When Archive and Restore Are Highest Priorities	41
What Is a Logical Volume Manager?	44

Chapter Overview

This chapter defines the terms and explains the concepts you need to understand to effectively perform the tasks described in Chapter 11, “Managing Disk Space.” In doing so, this chapter covers the following topics:

- Definitions of the physical and logical units **INFORMIX-OnLine Dynamic 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 first part of this chapter contains definitions for nine different units of disk storage: chunk, page, blobpage, extent, blobspace, dbspace, database, table, and tblspace. In addition to the strict definition of a given unit, the chapter also describes the role of each unit in storing and retrieving data. Noting the role or the purpose of a unit in addition to its strict definition facilitates your understanding of the units **OnLine** uses to store data.

Overview of Data Storage

OnLine can use two distinct types of disk space to manage physical disk I/O:

- Cooked file space, in which UNIX manages physical disk I/O
- Raw disk space, in which **OnLine** manages physical disk I/O

OnLine manages disk space using the following physical units:

- Chunk
- Page
- Blobpage
- Extent

Overlying the physical units of storage space, **OnLine** supports the following logical units associated with database management:

- Dbspace
- Blobspace
- Database
- Table
- Tblspace

OnLine maintains the following additional disk-space storage structures to ensure physical and logical consistency of data:

- Logical log
- Physical log
- Reserved pages

Because these additional disk-space structures are not permanent storage units, they are not described in this chapter. For information about the logical log, see Chapter 18, “What Is the Logical Log?” For information about the physical log, see Chapter 20, “What Is Physical Logging?” For information about reserved pages, see “Reserved Pages” on page 40-6.

The following sections describe the various data storage units **OnLine** supports, and the relationships between those units.

What Are the Physical Units of Storage?

OnLine uses the physical units of storage to allocate disk space. Unlike the logical units of storage whose size fluctuates, each of the physical units—chunks, extents, pages and blobpages—has a fixed or assigned size.

What Is a Chunk?

The *chunk* is the largest unit of physical disk dedicated to **OnLine** data storage. It represents an allocation of cooked disk space or raw disk space and is the *only* unit of physical storage allocated by the **OnLine** administrator. The **OnLine** administrator typically adds a chunk to a dbspace when that dbspace approaches full capacity. Figure 10-1 on page 10-5 illustrates how a chunk might appear if you choose to allocate space on a raw device. (See section “What Is a Raw Device?” on page 10-6.)

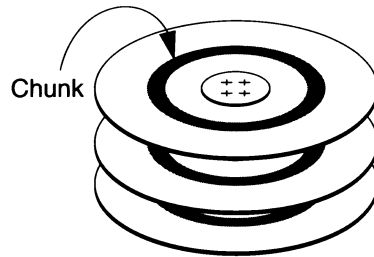


Figure 10-1 *A typical chunk allocated as raw disk space, represented by the darkened concentric circles on the platters of a disk drive*

In addition to providing administrators with a conveniently large unit for allocating disk space, **OnLine** also uses chunks for mirroring. A *primary chunk* is a chunk from which data is copied onto a *mirrored chunk*. If the primary chunk fails, the mirrored chunk is brought on-line automatically. For more information on mirroring, see Chapter 23, "What Is Mirroring?"

The maximum number of chunks that you can allocate for a given **OnLine** system is 2048. The maximum size of a chunk is 2 gigabytes.

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 if you use raw disk space, but this 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 a block device is configured with a character-special interface, the device is called a *raw device* and the storage space provided by that device 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, the character-special file can be distinguished from the block-special file by the first letter in the filename (typically “r”). For example, `/dev/rhd0f` is the character-special device corresponding to the `/dev/hd0f` 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 **OnLine** 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, **OnLine** manages the *internal* arrangement of data within the file. Never edit the contents of a cooked file managed by **OnLine** directly; to do so puts the integrity of your data at risk.

How Does OnLine 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 or a cooked device, the data is read 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 that if two users both read the password file, for instance, 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. **OnLine** requests that the data be placed in shared memory, which immediately makes it available to all **OnLine** 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 orders of magnitude better performance.

When you use a raw device to store your data, **OnLine** 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.)

If 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 the device

These steps are described in detail in “Allocating Raw Disk Space” on page 11-5.

Why Use a Cooked File?

Cooked files are easier to allocate 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 might sacrifice reliability and experience diminished performance if you store **OnLine** data in cooked files.

The buffering mechanism provided by most operating systems can produce 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 (that 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 created as an operating-system file, space in the chunk might not be physically contiguous.

Cooked files are less reliable than raw devices because I/O on a cooked file is managed by the UNIX operating system. 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, **OnLine** cannot guarantee that the committed data has actually reached the disk. **OnLine** recovery depends on the guarantee that data written to disk is actually on disk. If, in the event of system failure, the data is not present on disk, the **OnLine** automatic recovery mechanism might not be able to properly recover the data. (The data in the UNIX buffer might be lost completely.) 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 11-4.

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 10-31 for more information), you can subdivide partitions or cooked files into smaller chunks by 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 needed to reach a given chunk. For example, suppose that you create a 1,000 kilobyte chunk which 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 blob-space.

You might also need to specify an offset to prevent **OnLine** from overwriting partition information. “Do You Need to Specify an Offset?” on page 11-5 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 **OnLine** 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 OnLine Page Size” on page 11-13.) Because the size of your page is determined by your hardware, you cannot change page size. Figure 10-2 illustrates the concept of a page.

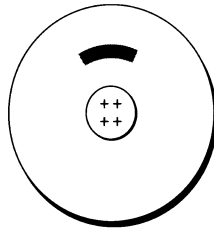


Figure 10-2 *A page, represented by a darkened sector of a disk platter*

A *chunk* is said to contain a certain number of pages, as illustrated in Figure 10-3. Note that a page is always entirely contained within a chunk; that is, a page cannot cross chunk boundaries.

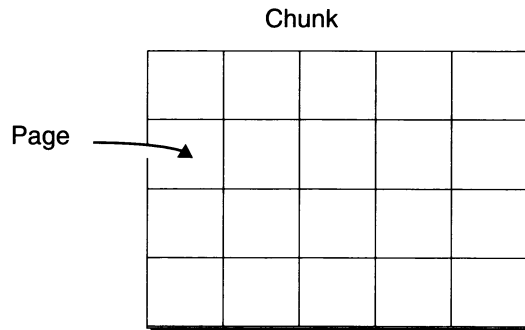


Figure 10-3 *A chunk, logically separated into a series of pages*

For information on how data within a page is structured, see Chapter 40, “OnLine Disk Structure and Storage.”

What Is A Blobpage?

A blobpage is the unit of disk-space allocation used by **OnLine** to store BYTE and TEXT data within a blobspace (See “What Is a Blobspace?” on page 10-19.) Blobpage size is specified as a multiple of the **OnLine** page size (See “Determining OnLine Page Size” on page 11-13). The **OnLine** administrator establishes the size of a blobpage when creating the blobspace in which it resides; the size of a blobpage can vary from blobpage to blobpage. Figure 10-4 illustrates the concept of a blobpage.

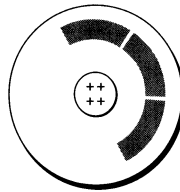


Figure 10-4 A blobpage, represented here as being a multiple (three) of a data page

Just as with pages in a chunk, a certain number of blobpages are said to compose a chunk in a blobspace, as illustrated in Figure 10-5. Note that a blobpage is always entirely contained in a chunk and cannot cross chunk boundaries.

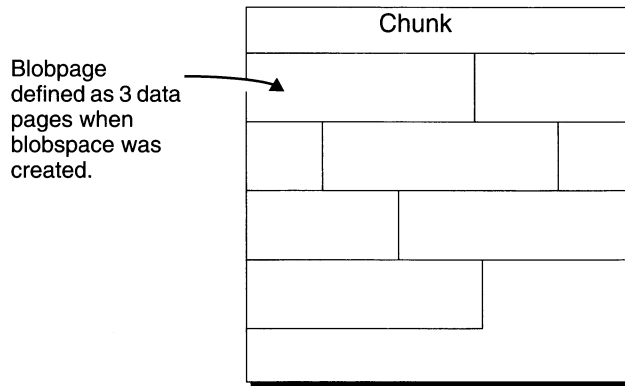


Figure 10-5 A chunk in a blobspace, logically separated into a series of blobpages

In addition to storing your blob data in a blob space, you can also choose to store your blob data in a db space. However, for blobs larger than two pages, you will find that performance improves when you store the blobs in a blob space. Blobs stored in a db space can share a page; blobs stored in a blob space do not share pages. For information about how data is structured when stored in a blob page, see “Structure of a Blob Space Blob Page” on page 40-60.

How Big Should a Blob Page Be?

When creating a blob space, you should aim to create a blob page size that approximates the size of the most frequently occurring blob to be stored within that blob space. For example, if you are storing 160 blobs and you expect 120 blobs to be 12 kilobytes and 40 blobs to be 16 kilobytes, a 12-kilobyte blob page size stores the blobs most efficiently. This configuration allows the majority (120) of the blobs to be stored using a single blob page, while the other 40 blobs require two blob pages each (with 8 kilobytes wasted in the second blob page).

However, there are circumstances in which you might want to use the larger, 16-kilobyte blob page size. If speed and reducing the number of locks are primary concerns, use a 16-kilobyte blob page so that every blob can be stored on a single blob page.

To continue the example, assume that your **OnLine** page size is 2 kilobytes. If you decide on a 12-kilobyte blob page size, specify the blob page size parameter as 6 (pages). If your **OnLine** page size is 4 kilobytes, specify the blob page size parameter as 3 (pages). In general, you divide the size of the blob (rounded up to the nearest kilobyte) by the page size to determine the blob page size parameter.

If a table has more than one blob column and the blobs are not close in size, store the blobs in different blob spaces, each with an appropriately sized blob page.

What Is an Extent?

When you create a table, **OnLine** allocates a fixed amount of space to contain the data to be stored in the table. When the table fills, **OnLine** must allocate more space for additional storage. The physical unit of storage that **OnLine** uses to allocate both the initial and subsequent storage space is called an *extent*. Figure 10-6 on page 10-12 illustrates the concept of an extent.

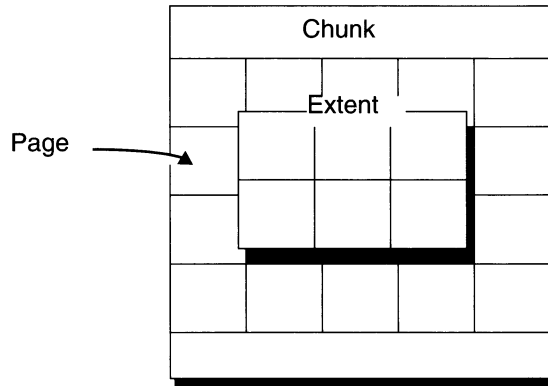


Figure 10-6 An extent consisting of size contiguous pages on a raw disk device

An extent consists of a collection of contiguous pages that store data for a given table. 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 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. You specify the initial extent size and next extent size using the CREATE TABLE and ALTER TABLE statements. See the *Informix Guide to SQL: Syntax* for more information.

Figure 10-7 illustrates the following key concepts concerning extent allocation:

- An extent is always entirely contained in a chunk; an extent cannot cross chunk boundaries.
- If OnLine cannot find available contiguous space in the first chunk equal to the size specified for the next extent (six pages in this case), it searches the next chunk in the dbspace for contiguous space.

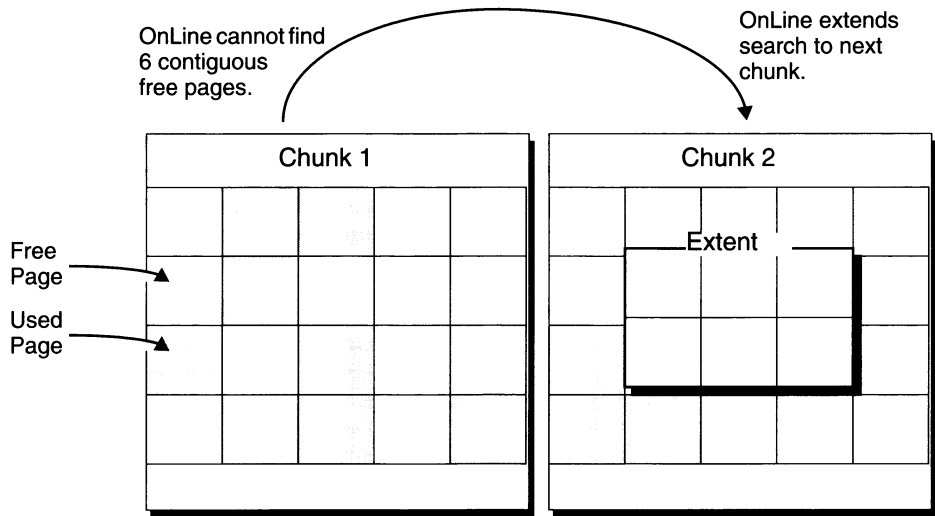


Figure 10-7 Process of extent allocation

Chapter 10 of the *Informix Guide to SQL: Tutorial* contains instructions on how to avoid the interleaving shown in chunk 1 of Figure 10-11.

What Are the Logical Units of Storage?

The logical units of **OnLine** storage fall into the following categories:

- Units of logical storage that function as accounting entities, including:
 - Dbspaces
 - Blobspace
 - Tblspaces
- Units of logical storage that are dictated by relational database design, including:
 - Databases
 - Tables

A *tblspace*, for example, does not correspond to any particular part of a chunk or even to any particular chunk. Instead, the indexes and data that make up a *tblspace* can be scattered throughout your chunks. The *tblspace*, however, represents a convenient accounting entity for space across chunks devoted to a particular table.

The following sections describe these logical storage units.

What Is a Dbspace?

A key responsibility of the **OnLine** administrator is to control where data is stored. 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 only affects 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 control in which physical units the data contained in logical storage units (such as database and table) resides.

How Can You Control Where Data Is Stored?

As Figure 10-8 shows, you control the placement of databases or tables using the *IN dbspace* option of the CREATE DATABASE or CREATE TABLE statements.

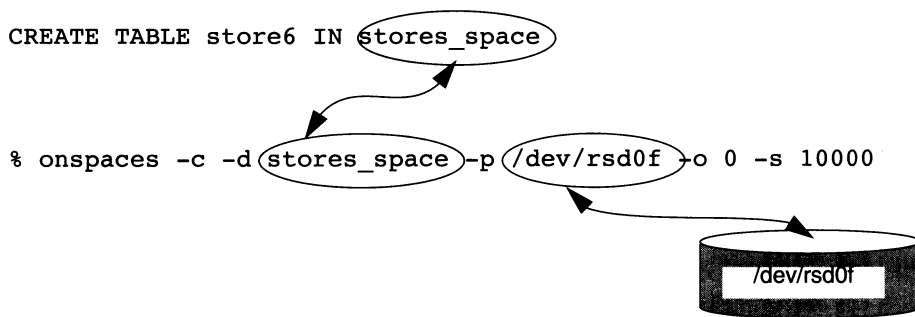


Figure 10-8 Controlling table placement using the CREATE TABLE... IN statement (Note that chronologically, you must issue the onspaces command before you create the table store6.)

Figure 10-8 also illustrates that before creating a database or table in a dbspace, you must first create the dbspace using **onspaces** or ON-Monitor. This is explained in “Creating a Dbspace Using onspaces” on page 11-9 and “Creating a Dbspace Using ON-Monitor” on page 11-9.

A dbspace is composed of one or more chunks, as shown in Figure 10-9. You can add more chunks at any time. As with blobspace chunks, it is a high-priority task of an **OnLine** administrator to monitor dbspace chunks for fullness and to anticipate the need to allocate more chunks to a dbspace. If a dbspace contains more than one chunk, you cannot control the chunk in which the data resides.

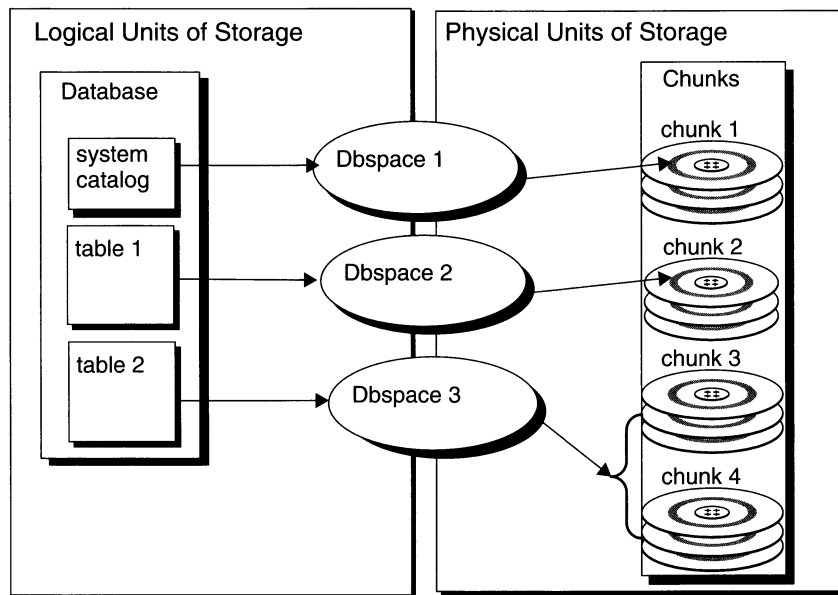


Figure 10-9 Dbspace provides a link between logical and physical units of storage

OnLine uses the dbspace to store databases and tables. You can store BYTE and TEXT data within a dbspace, but if the blobs are larger than two pages, performance can be poorer than if you stored the same data in a blobspace.

You must mirror every chunk in a mirrored dbspace. As soon as a mirror chunk is allocated, all space in the mirror chunk appears as full in the status displays output from **onstat -d** or the Dbspaces menu, Info option of ON-Monitor.

You can use ON-Monitor or **onspaces** to perform any of the following tasks related to dbspace management:

- Creating a Dbspace page 11-8
- Adding a Chunk to a Dbspace page 11-10
- Dropping a Dbspace or Blobspace page 11-15

You can use **onspaces** (but not ON-Monitor) to drop a chunk from a dbspace. See “Dropping a Chunk from a Dbspace Using onspaces” on page 11-14 for more information.

What Is the Root Dbspace?

The root dbspace is the initial dbspace created by **OnLine**. The root dbspace is special because it contains reserved pages and internal tables that describe and track all other dbspaces, blobspaces, chunks, databases, and tblspaces. (For more information on these topics see Chapter 40, “OnLine Disk Structure 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 **OnLine** is initialized.

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 **OnLine** 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 10-27 explains how much space you should allocate for the root dbspace. You can also add extra chunks to the root dbspace after you initialize **OnLine** disk space.

What Is a Temporary Dbspace?

A temporary dbspace is a dbspace reserved for the exclusive use of temporary tables. **OnLine** requires a temporary dbspace to store internal temporary tables generated by read-only queries on the secondary server when data replication is activated.

In addition, if temporary dbspaces exist and you list them in the `DBSPACETEMP` configuration parameter, **OnLine** uses them to store both implicit and explicit temporary tables. (See “What Is a Temporary Table?” on page 10-23.) Implicit temporary tables include tables **OnLine** creates when sorting, archiving, and performing warm restores. **OnLine** also uses temporary dbspaces to store explicit temporary tables created using the `WITH NO LOG` option, or when a nonlogging database is the current database.

Unlike a temporary table, **OnLine** never drops a temporary dbspace unless explicitly directed to do so. A temporary dbspace is only temporary in the sense that none of its contents are preserved should **OnLine** shut down abnormally. Temporary dbspaces are designed for the exclusive storage of temporary tables.

All temporary dbspaces are reinitialized whenever you initialize **OnLine**. This means that **OnLine** clears any tables that might have been left over from the last time that **OnLine** shut down.

Both logical and physical logging are suppressed for temporary dbspaces. Temporary dbspaces are never archived as part of a full-system archive. You cannot mirror a temporary dbspace.

You can also use temporary disk space to improve disk-load balancing, as explained in “Spread Your Temporary Storage Space Across Multiple Disks” on page 10-34.

For detailed instructions on how to create a temporary dbspace with `ON-Monitor` or `onspaces`, see “Creating a Dbspace” on page 11-8.

What Are the Advantages of Using Temporary Dbspaces?

OnLine suppresses logical logging for implicit temporary tables and explicit temporary tables created with the `WITH NO LOG` options that reside in a temporary dbspace. For a temporary table in a standard dbspace, inserts and updates are not logged. However, **OnLine** does log table creation, the allocation of extents, and the dropping of the table. Some correlated subqueries that create and drop a temporary table for each row in a master table can generate a large amount of log data if these temporary tables do not have a temporary

dbospace in which to reside in. Logical-log suppression in temporary dbospaces reduces the number of log records to roll forward during logical recovery as well, thus improving the performance during critical down time.

All physical logging is suppressed in temporary dbospaces. This 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 that can have a negative impact on performance. When temporary tables reside in temporary dbospaces, operations on the temporary tables are not physically logged, thus necessitating fewer checkpoints.

Using temporary dbospaces also reduces the size of your archive. Because **OnLine** does not archive temporary dbospaces, the time required for archiving and restoring is reduced when you use temporary dbospaces to store temporary tables.

What Is a Blobspace?

A blobspace is a logical storage unit composed of one or more chunks that store only BYTE and TEXT data. A blobspace stores BYTE and TEXT data in the most efficient way possible. Blobs associated with distinct tables can be stored within the same blobspace. Blob data stored in a blobspace is written directly to disk and 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, blobs stored in a blobspace are not written to either the logical or physical log. The blobspace blobs are logged by writing the blobs directly from disk to the logical-log backup tapes when logical logs are archived. Blobspace blobs never pass through the logical-log files.

When you create a blobspace, you assign to it one or more chunks. You can add more chunks at any time. One of the tasks of an **OnLine** administrator is to monitor the chunks for fullness and anticipate the need to allocate more chunks to a blobspace. See "Monitoring Blobs in a Blobspace" on page 29-53 for instructions on how to monitor chunks for fullness. See "Creating a Blobspace" on page 11-12 for instructions on how to create a blobspace.

You can designate one or more mirrored blobspaces; blobspaces that are mirrored require one mirror chunk for each primary chunk.

For information about the structure of a blobspace, see "Structure of a Blob-space" on page 40-54.

The following list contains tasks you can perform related to blob space management. You can use ON-Monitor or **onspaces** to perform any of the tasks listed.

- Creating a Blob space page 11-12
- Adding a Chunk to a Blob space page 11-14
- Dropping a Dbspace or Blob space page 11-15

You can use **onspaces** (but not ON-Monitor) to drop a chunk from a dbspace. See “Dropping a Chunk from a Blob space” on page 11-15 for more information.

What Is a Database?

A database is a logical storage unit that contains tables 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 10-10 on page 10-21 shows the **stores6** database.

A database resides in the dbspace named in the SQL statement CREATE DATABASE. If no dbspace is specified, the database resides in the root dbspace. The implications of stating that a database is located in a dbspace are as follows:

- The database system catalog tables are stored in that dbspace.
- That dbspace is the default location of tables not explicitly created in other dbspaces.

You can spread the tables of a database across multiple dbspaces and, consequently, multiple devices. Doing so can have a significant impact on performance, as explained in “How Much Disk Space Do You Need to Store Your Data?” on page 10-27.

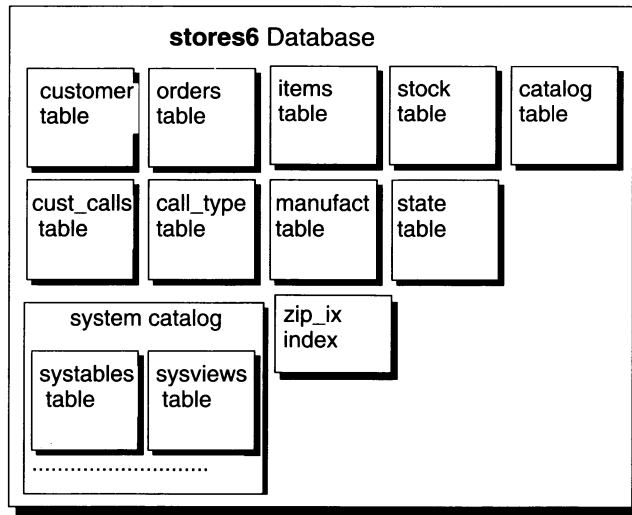


Figure 10-10 The stores6 database

See “How Much Disk Space Do You Need to Store Your Data?” on page 10-27 for advice on where to put your databases.

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. This also limits the size of the database to the size of the chunk.

See “OnLine Disk Structure and Storage” on page 40-3 for instructions on how to list the databases that you create.

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, **OnLine** allocates disk space for the table in a block of pages called an extent. (See “What Is an Extent?” on page 10-11.) You can specify the size of both the first and any subsequent extents. (See Chapter 10 of the *Informix Guide to SQL: Tutorial* for instructions.)

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). If the user does not specify the dbspace, the table is placed in the dbspace where the database resides.

A table resides completely in the dbspace in which it was created. The **OnLine** 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. "How Can You Control Where Data Is Stored?" on page 10-15 explains how to place a table in a given dbspace using the *IN dbspace* option of the CREATE TABLE statement.

A table, composed of extents, can span multiple chunks, as shown in Figure 10-11.

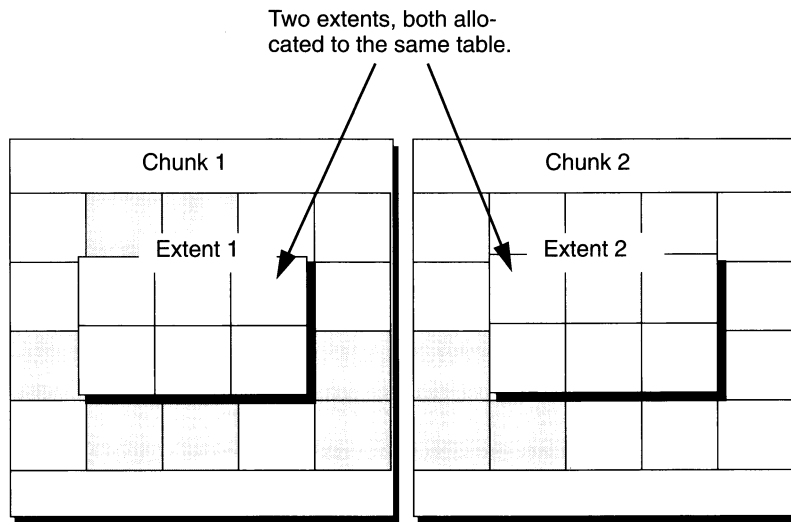


Figure 10-11 Table spanning more than one chunk

Blob data associated with a table can reside either in the dbspace with the rest of the table data or in a separate blob space. If you are using **INFORMIX-OnLine/Optical**, you can also store blobs in an optical storage subsystem.

For advice on where to store your tables, see "Isolate High-Use Tables" on page 10-31 and also Chapter 10 of the *Informix Guide to SQL: Tutorial*.

What Is a Temporary Table?

There are two types of temporary tables: *explicit* temporary tables and *implicit* temporary tables.

An *explicit* temporary table is a temporary table that you create using the TEMP TABLE option of the CREATE TABLE statement or the INTO TEMP clause of the SELECT statement. For instance, the following SQL statement explicitly creates a temporary table:

```
SELECT * FROM customer INTO TEMP temp_table
```

When an application creates an explicit temporary table, it exists until the application takes one of the following actions:

- The application terminates
- The application closes the database in which the table was created and opens a database in a different database server.
- The application closes the database in which the table was created. (In this case, the table is dropped only if the database does transaction logging and the temporary table was not created with the NO LOG option).

When either of these three events occurs, the temporary table is deleted.

An *implicit* temporary table is a temporary table created by **OnLine** 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

An implicit temporary table is deleted when the processing that initiated the creation of the table is complete.

If **OnLine** shuts down without adequate time to clean up temporary tables, it performs temporary table cleanup as part of the next initialization. (To request shared-memory initialization without temporary table cleanup, execute **oninit** with the **-p** option.)

Where Are Temporary Tables Stored?

The **dbspace** in which **OnLine** stores temporary tables depends on whether the table is an explicit or implicit table. Both cases are examined in detail in following two sections.

Explicit Temporary Tables

If you create an explicit temporary table using the **IN *dbspace*** option of **CREATE TEMP TABLE**, the temporary table is stored in that **dbspace**.

If you do not use the **IN *dbspace*** option of **CREATE TEMP TABLE** or if you create the explicit table with **SELECT INTO TEMP**, **OnLine** checks the **DBSPACETEMP** environment variable and the **DBSPACETEMP** configuration parameter. (The environment variable supersedes the configuration parameter.) If **DBSPACETEMP** is set, **OnLine** stores the explicit temporary table in one of the **dbspaces** specified in the list.

OnLine keeps track of which was the last **dbspace** in the list that it used to store a temporary table. When **OnLine** receives another request for temporary storage space, it uses the next **dbspace** in the list. In this way, **OnLine** spreads I/O evenly across temporary storage space you specify in **DBSPACE-TEMP**. If **OnLine** finds that you do not specify any temporary **dbspaces** in **DBSPACETEMP**, or the temporary **dbspaces** that you specify have insufficient space, it creates the table in a standard (nontemporary) **dbspace**.

By default (you did not specify a **dbspace** using the **IN *dbspace*** option of **CREATE TEMP TABLE** and the **DBSPACETEMP** is not set), **OnLine** takes the following actions:

- If you created the temporary table with **CREATE TEMP TABLE**, it is stored in the same **dbspace** in which the database that contains the temporary table resides.
- If you created the temporary table with the **INTO TEMP** option of the **SELECT** statement, it is stored in the root **dbspace**.

Implicit Temporary Tables

OnLine stores implicit temporary tables in one of the dbspaces you specify in the DBSPACETEMP environment variable or the DBSPACETEMP configuration parameter. (The environment variable supersedes the configuration parameter.) If DBSPACETEMP is not set, **OnLine** stores the temporary table in the root dbspace.

When **OnLine** creates temporary implicit 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 see "Specifying Where Sorting Occurs" on page 30-7.

What Is a Tblspace?

OnLine administrators sometimes need to track disk usage by a particular table. A logical unit of storage called *tblspace* that contains all the disk space allocated to a given table facilitates this tracking.

The tblspace contains the following types of pages:

- Pages allocated to data
- Pages allocated to indexes
- Pages used to store blob data in the dbspace (but not pages used to store blob data in a blob space)
- Bit-map pages that track page usage within the table extents

Figure 10-12 on page 10-26 illustrates the tblspaces for three tables that form part of the **stores6** database. There is one and only one table per tblspace.

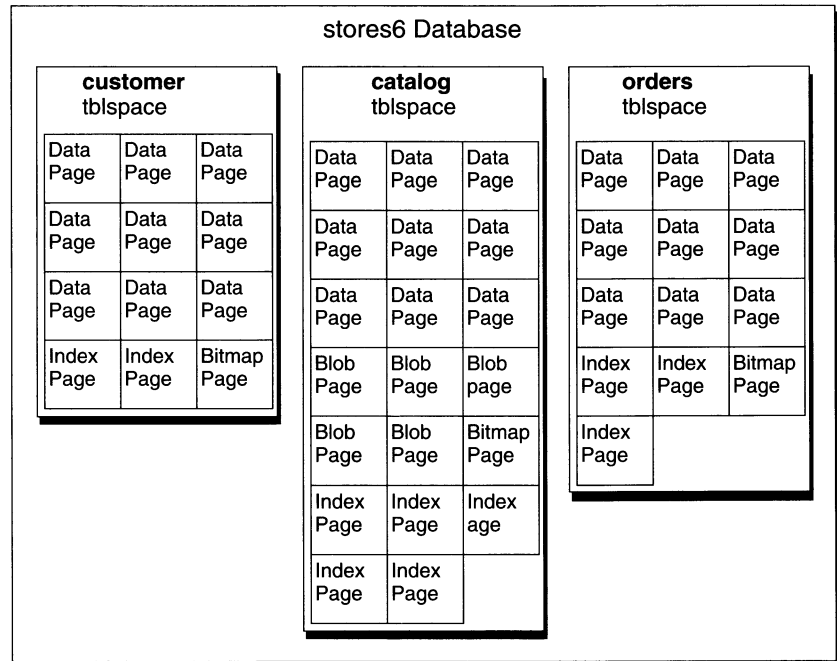


Figure 10-12 Three of the sample tblspaces in stores6 database. Blob pages represent blob data stored in a dbspace.

What Is Extent Interleaving?

The pages that belong to a tblspace are allocated 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 10-13 on page 10-27 depicts this situation with two noncontiguous extents belonging to the tblspace for **table_1** and a third extent belonging to the tblspace for **table_2**. A **table_2** extent is positioned between the first **table_1** extent and the second **table_1** extent, causing the **table_1** extents to be noncontiguous. When this situation occurs the extents are said to be interleaved. Because sequential access searches across **table_1** require the disk head to seek across the **table_2** extent, performance is poorer than if the **table_1** extents were contiguous. See Chapter 10 of the *Informix Guide to SQL: Tutorial* for instructions on how to avoid and eliminate interleaving of extents.

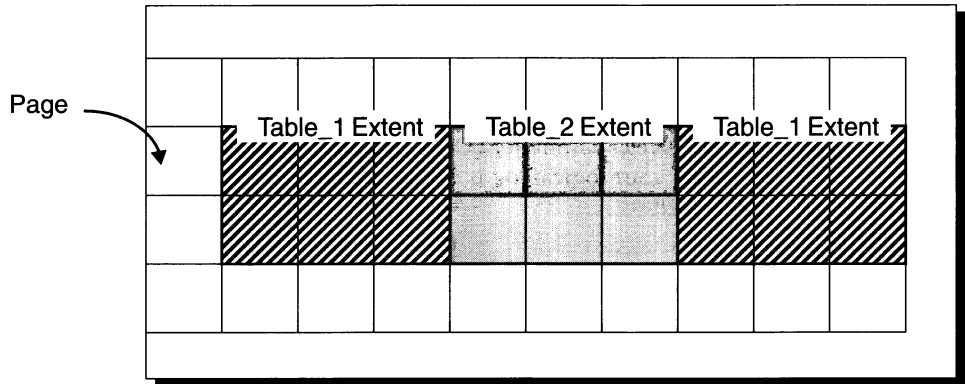


Figure 10-13 Three extents belonging to two different tablespaces in a single database space

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:

- Calculate the size requirements of the root database space
- Estimate the total amount of disk space to allocate to all **Online** databases, including space for overhead and growth

These steps are explained in the following sections.

Calculate the Size of the Root Database Space

To calculate the size of the root database space, you must take the following storage structures into account:

- The physical and logical-log files
- Temporary tables
- Data
- Online-Archive catalog data
- Control information

Each of these factors is considered separately in the sections that follow.

Physical and Logical Logs

The size of your physical log is defined by the value stored in the ONCONFIG parameter PHYSFILE. Advice on sizing your physical log is contained in “How Big Should the Physical Log Be?” on page 20-5.

To calculate the size of the logical-log files, multiply the value of the ONCONFIG parameter LOGSIZE by the number of logical-log files. Advice on sizing your logical log is contained in “Logical-Log Size Guidelines” on page 18-6.

Temporary Tables

Analyze end-user applications to estimate the amount of disk space that **OnLine** might require for implicit temporary tables. “What Is a Temporary Table?” on page 10-23 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 returned provides a good basis for estimating the amount of space required.

OnLine creates implicit temporary files when you use ON-Archive to perform a warm restore. The largest implicit temporary file that **OnLine** creates during a warm restore is equal to the size of your logical log. You calculate the size of your logical log by multiplying the value of LOGSIZE by LOGFILES. For more information on these configuration parameters, see “What Should Be the Size and Number of Logical-Log Files?” on page 18-7.

You must also analyze end-user applications to estimate the amount of disk space that **OnLine** might require for explicit temporary tables. (See “What Is a Temporary Table?” on page 10-23.)

By default, both implicit and explicit temporary tables are stored 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 that **OnLine** uses to store temporary files and tables. See “Where Are Temporary Tables Stored?” on page 10-24.

Data

Next, decide if users are to store databases or tables in the root dbspace. If the root dbspace is the only dbspace 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 archive your data and perform logical-log backups, you should include space estimates for ON-Archive catalog data. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for a description of the tables that compose the ON-Archive catalog. See the *Informix Guide to SQL: Tutorial* for instructions on how to calculate the size of each table.

Control Information (Reserved Pages)

The total amount of disk space required for **OnLine** control information is 3% of the size of the root dbspace (sum of physical and logical log, temporary space, and data) plus 14 pages, expressed as kilobytes (or 14 x **OnLine** page size).

Complete the Root Dbspace Calculation

Now calculate the size of the root dbspace, adding the following values for a root dbspace size:

1. Physical log
2. Logical log
3. Disk space for temporary tables
4. Disk space for data stored in the root dbspace
5. Disk space for the reserved pages
6. Disk space to accommodate ON-Archive catalog data, if you use ON-Archive for performing archives and logical-log file backups

You need not store the physical log, the logical log, nor temporary tables in the root dbspace. Only include calculations for these 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 rootdbs might differ markedly from the final configuration; you can resize the rootdbs after you remove the physical and logical logs. However, the rootdbs must be large enough for the minimum size configuration during disk initialization.

Estimate Space Required by Databases Including Overhead and Growth

The amount of additional disk space needed for **OnLine** data storage depends on the needs of your end-users. Every application that your end-users run has different storage requirements. The following list suggests some of the steps you might take to help you 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.

Refer to Chapter 10 of the *Informix Guide to SQL: Tutorial* for instructions about calculating the size of your tables.

Disk-Layout Guidelines

The following goals for efficient disk layout are typical in a production environment:

- Limiting disk head movement
- Reducing disk contention
- Balancing the load
- Maximizing availability

The following sections describe strategies that you can use to achieve these goals and present some sample disk layouts. Each sample disk layout illustrates a disk-organization scheme suited to a certain set of requirements, resources, and priorities for data storage.

Strive to Associate Partitions with Chunks

When you allocate disk space (raw disk or cooked files), you allocate it in chunks. A dbspace or a blobspace is associated with one or more chunks. You must allocate at least one chunk for the root dbspace.

Informix recommends that you format your disks so that each chunk is associated with its own UNIX disk partition. When every chunk is defined as a separate partition (or device), it is easy to track disk-space usage; you can avoid errors caused by miscalculated offsets.

A disk that is already partitioned might require the use of offsets. See “Do You Need to Specify an Offset?” on page 11-5 for details.

Consider Mirroring

You can mirror critical tables and databases to maximize availability. You should mirror the root dbspace, the dbspace that contains the physical log, and the dbspace that contains the logical-log files. You specify mirroring on a chunk by chunk basis. Locate the primary and the mirrored chunk on different disks. Ideally, different controllers should handle the different disks. Figure 10-14 shows a primary chunk and its mirror.

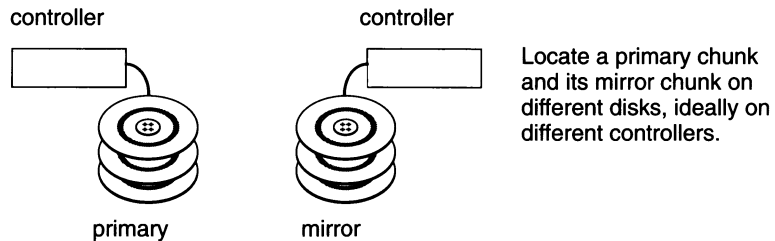


Figure 10-14 A primary chunk and its mirror

Isolate High-Use Tables

You can place a table with high I/O activity on a disk device dedicated to its use and thus reduce contention for the data stored in the table. When disk drives have different performance levels, you can put the tables with the highest frequency of use on the fastest drives. Placing two high-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 table with the high frequency of use in the dbspace just created using the `IN dbspace` option of `CREATE TABLE`. Figure 10-15 illustrates this strategy by showing optimal placement of three tables with high frequency of use.

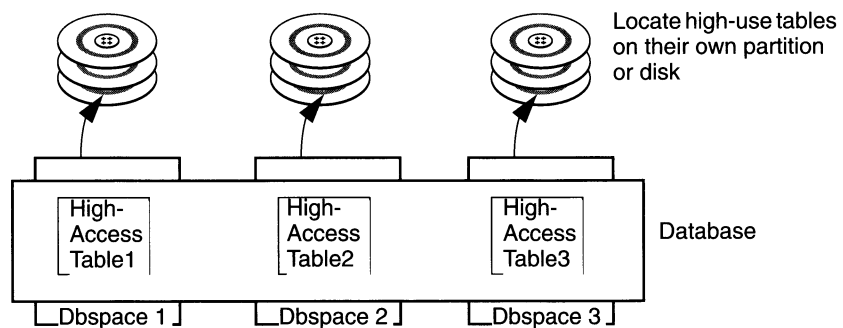


Figure 10-15 Example of isolation of high-use tables. Each table is located in a different dbspace.

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`. For details, see “-g Monitoring Options” on page 37-55.

Group Your Tables with Archive and Restore in Mind

When deciding 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. This might influence which tables you group together in a particular dbspace.

Although you must perform a cold restore if a dbspace containing critical media fails, you need only perform a warm restore if a non-critical dbspace fails. This might influence the dbspace you use to store critical media. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for more information.

Spread a Single Table Across Multiple Disk Devices to Reduce Contention

To reduce contention between different programs using the same table, you can also spread the table across multiple devices. To do this, put the table in a dbspace that includes multiple chunks, each of which is located on a different disk. Although you have no control over how the table data is spread across the chunks, this layout might result in multiple disk access arms for one table. Figure 10-16 shows a table spread across multiple tables.

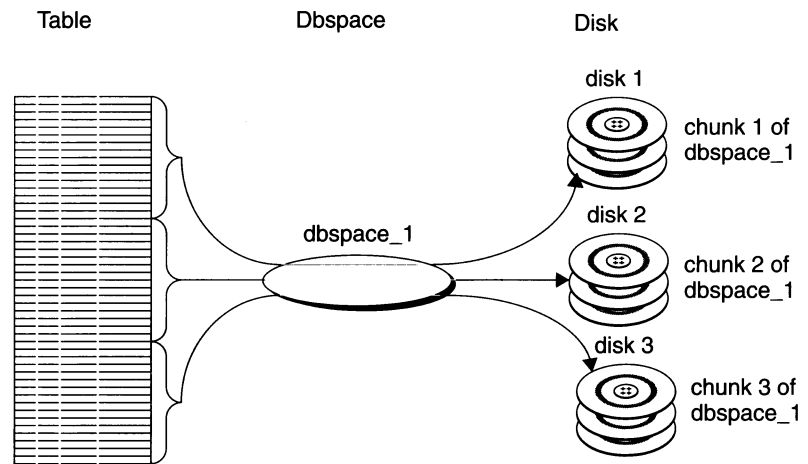


Figure 10-16 Table spread across multiple disk devices. Dotted lines indicate that actual storage location of a given row cannot be predetermined.

Place High-Use Tables on Middle Partition of Disk

To minimize disk-head movement, place the most frequently accessed data in partitions as close to the middle of the disk as possible (see Figure 10-17). (When a disk device is partitioned, the middlemost 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.

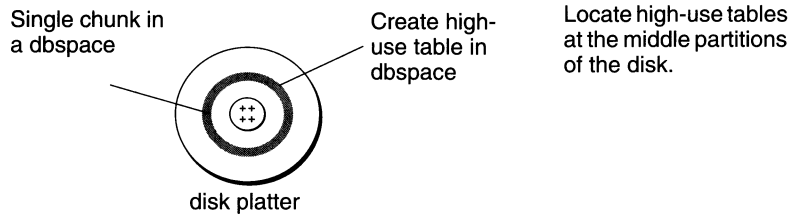


Figure 10-17 *Illustration of disk platter with high-use table located on middle partitions of a disk*

To place high-use 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-use tables, use the `IN` option of `CREATE TABLE` to place them in the newly created dbspace.

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 nontemporary dbspaces. By designing the list in such a way that your temporary disk space is spread across multiple disks, you achieve load balancing. For instructions on how to set the `DBSPACETEMP` configuration parameter see “`DBSPACETEMP`” on page 35-13.

Optimize Table Extent Sizes

As explained in “What Is a Tblspace?” on page 10-25, when two or more large, growing tables share a dbspace, their new extents might be interleaved. This interleaving creates gaps between the extents of any one table. (See Figure 10-18 on page 10-35.) 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. See Chapter 10 in the *Informix Guide to SQL: Tutorial* for advice on how to alleviate this problem. You can also consider placing the tables in separate dbspaces.



Figure 10-18 Illustration of interleaved extents

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 files that **OnLine** accesses frequently. Likewise, reserved pages are also accessed frequently; they contain internal tables that describe and track all dbspaces, blobspaces, chunks, databases, and tblspaces.

By default, the logical and physical log are stored together with the reserved pages in the root dbspace. Although this is convenient if you have a small, low-volume transaction-processing system, maintaining these files together in the root dbspace can become a source of contention as your database system grows.

You can reduce this contention and provide a form of load balancing by moving 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, or preferably, consider storing them in a dbspace dedicated to storing only the physical or logical log. See “Where Is the Physical Log Located?” on page 20-7 and “Where Should Logical-Log Files Be Located?” on page 18-8 for more advice on where to store your logs.

For instructions on how to change the location of the logical and physical log, see “Changing the Physical-Log Location and Size” on page 21-3 and “Moving a Logical-Log File to Another Dbspace” on page 19-6.

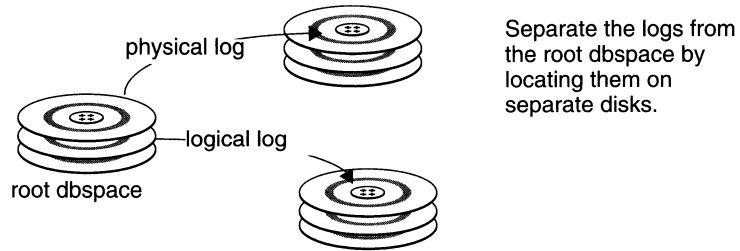


Figure 10-19 Separation of root dbspace, physical log and logical log onto different disk drives

Take into Account Archive and Restore Performance

When planning your disk layout, consider how the configuration you choose affects your archive and restore procedure. This section describes two configurations that can have a significant impact on your archive and restore procedure.

Cluster Catalogs with the Data They Track

When a disk that contains the system catalog for a particular database fails, the entire database remains inaccessible until the catalog is restored. Because of this, 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 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 from one another, this configuration is also the least desirable in terms of recovery.

Whenever a disk that contains critical media (the root dbspace, physical log, and logical log) fails, **OnLine** comes off-line. In addition, the **OnLine** administrator must restore all **OnLine** data, starting in off-line mode, from a level-0 archive before processing can continue.

By separating 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 media (either the root dbspace, physical log, or logical log). "Sample Layout When Archive and Restore Are Highest Priorities" on page 10-41 explains this concept in detail.

Sample Disk Layouts

This section describes three sample disk layouts. It presents concrete examples of approaches an **OnLine** administrator might use to apportion disk space given a certain set of needs, priorities, and resources.

The setting for the sample disk layouts is a fictitious sporting-goods distributor that uses the structure (but not the volume) of the **stores6** database. In this example, the **OnLine** database server is configured to handle approximately 350 users and 6 gigabytes of data.

Storage for the system requires two large, high-access tables: **cust_calls** and **items**. Assume that both of these tables contain more than 1 million rows and are subject to constant access from users around the country.

The **cust_calls** table represents a record of all customer calls made to the distributor. The **items** table is a table containing a line item of every order ever shipped by the distributor. The remaining tables are low- or moderate-volume tables used by **OnLine** to look up data such as zip-code or manufacturer.

When setting out to organize disk space, an **OnLine** administrator usually has one or more of the following objectives in mind:

- High performance
- High availability
- Ease of archive and restore

There are trade-offs involved in meeting any one of the three objectives. For example, configuring your system for high performance usually results in taking risks regarding the availability of data and ease of archive and restore. The sections that follow describe these trade-offs in detail and explain strategies you can use to achieve each of these three objectives.

The resources available limited the distributor to the purchase of four disk drives, two with a storage capacity of two gigabytes and two with a storage capacity of one-and-a-half gigabytes. In addition to the extra half-gigabyte storage capacity, the two-gigabyte drives also delivered better performance than the one-and-a-half gigabyte drives.

Only in the case when availability is highest priority is it assumed that the **OnLine** administrator can acquire another disk drive for mirroring.

Sample Layout When Performance Is Highest Priority

This section describes the strategies used by the **OnLine** administrator when performance is the first priority.

Strategies Used to Achieve High Performance

Three strategies are used to ensure high performance.

First, the administrator isolates the highest-use tables, **cust_calls** and **items**, in separate dbspaces reserved solely for these tables, as shown in Figure 10-20.

Second, the administrator places those two tables on the fastest drives (disk drives 2 and 3 in Figure 10-20).

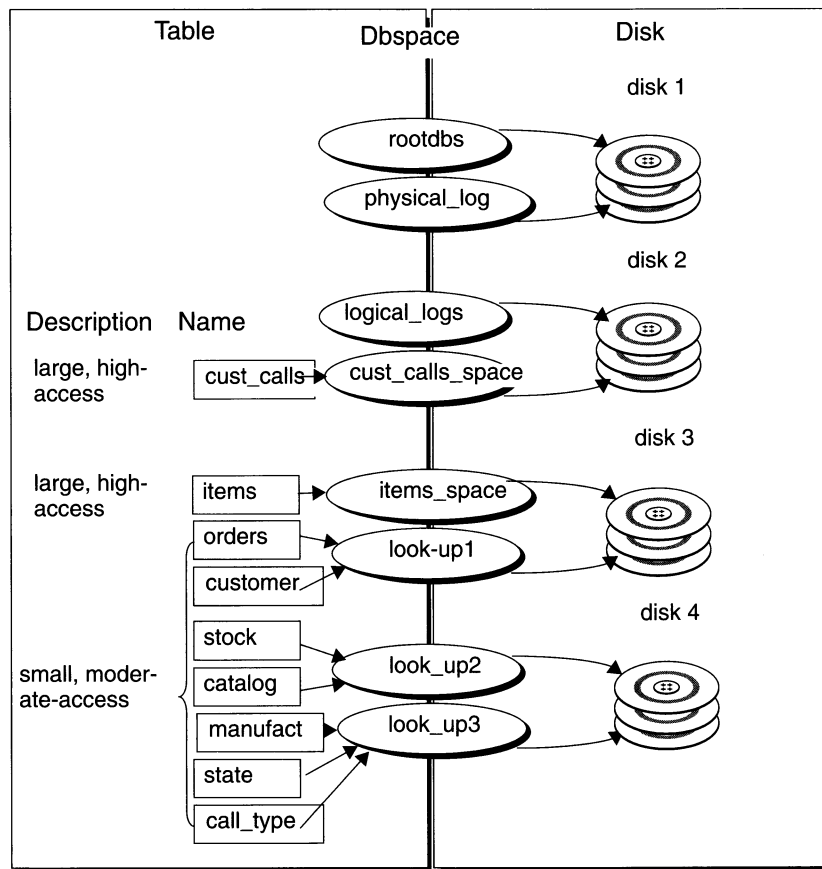


Figure 10-20 Disk layout when performance is highest priority

Third, the physical and logical logs are separated from each other by placing them in separate dbspaces.

Implementing the Strategies

To implement the first two strategies, the administrator creates the **cust_calls_space** and **items_space** dbspaces on the fastest drives using the **-c** option of **onspaces**.

The administrator then uses the **IN *dbspace*** option of **CREATE TABLE** to create and place the **cust_calls** and **items** tables as shown in the following example:

```
CREATE TABLE cust_calls IN cust_calls_space
CREATE TABLE items IN items_space
```

To implement the third step, the administrator uses the **onparams** utility to drop the logical-log files from the root dspace. The administrator then adds the logical-log files to the **logical_logs** dspace, using **onparams**. (See “Moving a Logical-Log File to Another Dspace” on page 19-6.)

To move the physical log from the root dspace to the **physical_log** dspace, the administrator again uses the **onparams** utility as described in “Using onparams to Change Physical-Log Location or Size” on page 21-5.

Sample Layout When Availability Is Highest Priority

This scenario is identical to the high-performance scenario except that here the **OnLine** administrator acquires an additional disk drive and places a high priority on keeping the database server on-line.

Strategies Used to Ensure Maximum Availability

The surest way to maximize the availability of data is to mirror all dbspaces managed by **OnLine**. Because just one disk drive is available for mirroring, the **OnLine** administrator uses it to mirror critical media (root dbspace, physical log, and logical log). Although this strategy does not guarantee the constant availability of all data, it does ensure that **OnLine** remains in on-line mode even if critical media fails.

Implementing the Strategies

As you can see in Figure 10-21, the **OnLine** administrator allocates three new mirror chunks on disk 5, one for each primary chunk in the **rootdbs**, **physical_log**, and **logical_logs** dbspaces.

The administrator then adds one of the chunks to each dbspace and starts mirroring using the **-m** option of **onspaces**. (See “Starting Mirroring” on page 24-5 for detailed instructions on how to initiated mirroring.)

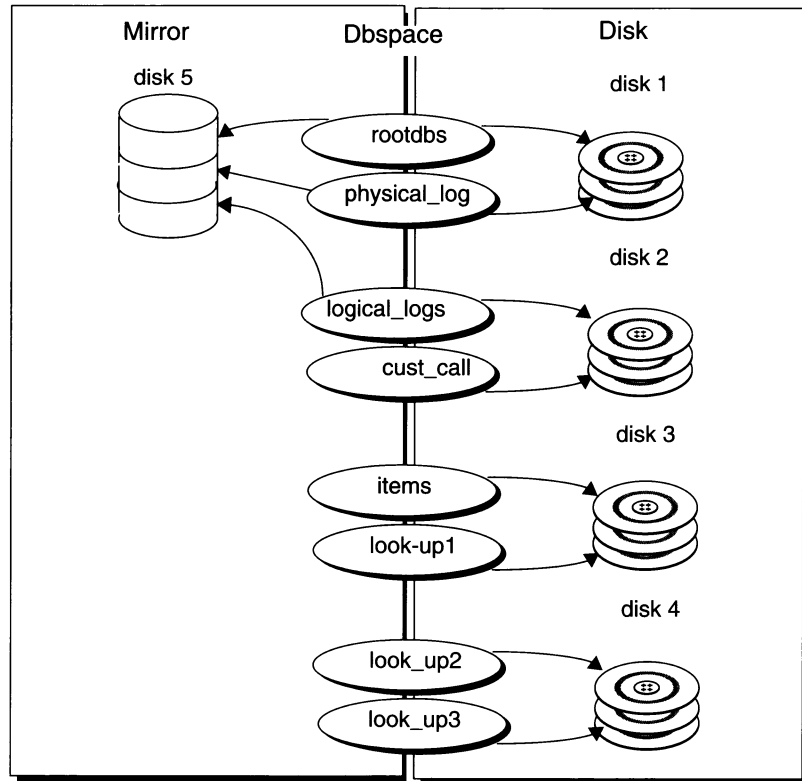


Figure 10-21 Disk layout when availability is highest priority

Sample Layout When Archive and Restore Are Highest Priorities

The following section describes a sample disk layout when ease of archive and restore is highest on an **OnLine** administrator's list of priorities. The configuration described assumes the same data and resources that were available in the first sample layout when the administrator's highest priority was performance.

Strategies Used to Achieve Ease of Archive and Restore

In "Sample Layout When Performance Is Highest Priority" on page 10-37 the root dbspace, the physical-log, and the logical-log files are spread across three disks, thus maximizing performance of the database server in terms of stor-

age and retrieval. (See Figure 10-22 on page 10-43.) However, if a disk fails, chances are three in four that it might be one of the disks that contains critical media (the root dbspace, physical log, or logical log). The failure of any of these disks would bring **OnLine** down and necessitate a time-consuming cold restore from a level-0 archive.

The configuration shown in Figure 10-22 on page 10-43 depicts the disk layout implemented by an **OnLine** administrator for whom availability and ease of archive and restore are of the highest priority. Here, the root dbspace, physical log, and logical-log files are clustered together on the same disk.

Although this increases disk contention for access to these logs and the root dbspace, in the event of a disk failure chances are only one in four that the disk containing critical media (the root dbspace, physical log, and logical log) would be the disk that fails. If one of the other disks should fail, the **OnLine** administrator can perform a warm restore of the affected dbspace while the database server remains on-line.

Implementing the Strategies

By default, the physical and logical log are located in the root dbspace, so no action is required on the part of the system administrator to implement the ease of archive and restore strategy.

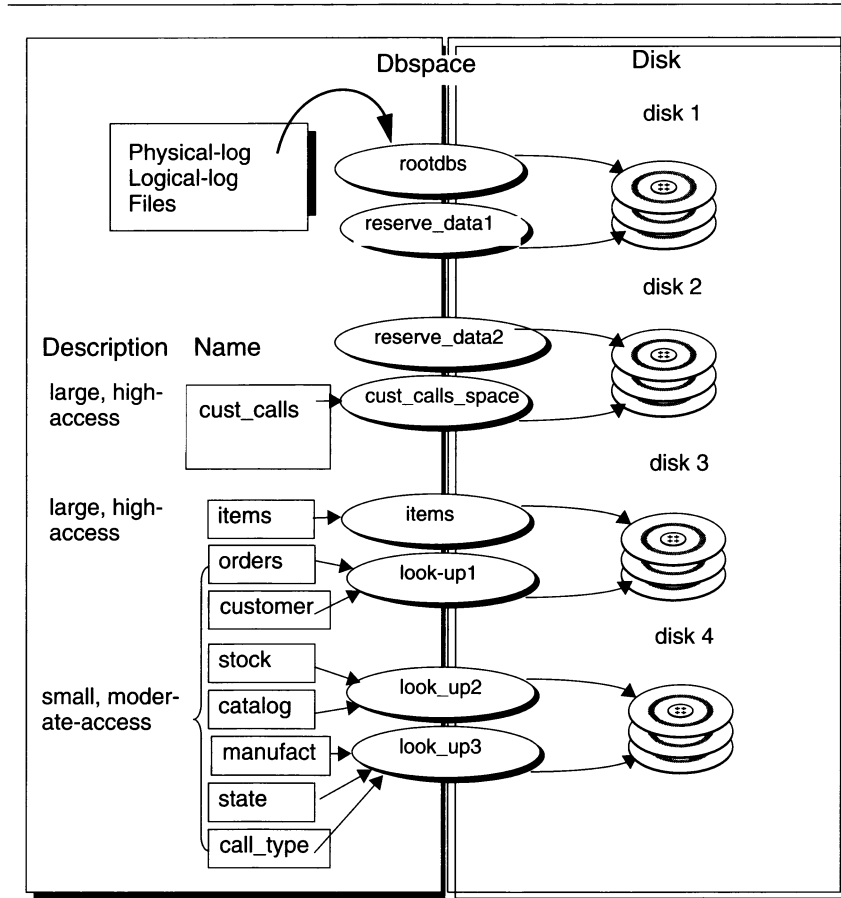


Figure 10-22 Disk layout when ease of archive and restore is highest priority

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 **OnLine** 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. **OnLine** chunks are limited to a size of two gigabytes and this size can only be attained 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 two 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 consisting of the middle-most cylinders of a disk drive and placing high-use tables in that volume. See “Place High-Use Tables on Middle Partition of Disk” on page 10-33 for more information. (Technically, you do not place a table directly in a volume, but must first allocate a chunk as a volume, then assign the chunk to a dbspace, and finally place the table in the dbspace. See “How Can You Control Where Data Is Stored?” on page 10-15 for more information.)

You can also improve performance by using a logical volume manager to define a volume that spreads across multiple disks, and then place a table in that volume. This helps reduce contention between programs that access the same table, as explained in “Spread a Single Table Across Multiple Disk Devices to Reduce Contention” on page 10-33.

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. This means that 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. See “What Mirroring Alternatives Exist?” on page 23-5 for more information.

Managing Disk Space

Chapter Overview	3
Allocating Disk Space	3
Allocating Cooked File Space	4
Allocating Raw Disk Space	5
Do You Need to Specify an Offset?	5
Creating Links to Each Raw Device	6
Initializing Disk Space	7
Initializing Disk Space with ON-Monitor	7
Initializing Disk Space with oninit	7
Creating a Dbspace	8
Creating a Dbspace Using ON-Monitor	9
Creating a Dbspace Using onspaces	9
Adding a Chunk to a Dbspace	10
Adding a Chunk	10
Adding a Chunk Using ON-Monitor	11
Adding a Chunk Using onspaces	11
Creating a Blobospace	12
Determining OnLine Page Size	13
Creating a Blobospace Using ON-Monitor	13
Creating a Blobospace Using onspaces	14

Adding a Chunk to a BlobSpace	14
Dropping a Chunk from a DbSpace Using onspaces	14
Dropping a Chunk from a BlobSpace	15
Dropping a DbSpace or BlobSpace	15
Dropping a DbSpace or BlobSpace Using ON-Monitor	16
Dropping a DbSpace or BlobSpaces Using onspaces	16
Optimizing BlobSpace Blobpage Size	16
Determining BlobSpace Storage Efficiency	17
BlobSpace Storage Statistics	17
Determining Blobpage Fullness with oncheck -pB	17
Interpreting Blobpage Average Fullness	19
Apply Efficiency Criteria to Output	19
Managing Extents	19
If You Find That Extents Are Interleaved	20
Managing Tables	21
Reclaiming Space in an Empty Extent Using Alter Index	21
Reclaiming Space in an Empty Extent Using the UNLOAD and LOAD Statements	21

Chapter Overview

This chapter provides the instructions you need to effectively manage disk space and data controlled by **INFORMIX-OnLine Dynamic Server**. It assumes you are familiar with the terms and concepts contained in Chapter 10, "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, chunks and dbspaces
 - Allocating, adding and dropping chunks from dbspaces and blobspaces
 - Creating and dropping dbspaces and blobspaces
- How to optimize blobpage size
- How to reclaim space in an empty extent

Allocating Disk Space

This section explains how to allocate disk space for **OnLine**. Before you allocate disk space you should read the following sections:

- "Should You Allocate Chunks as Cooked Files or Raw Disk Space?" on page 10-5
- "How Much Disk Space Do You Need to Store Your Data?" on page 10-27
- "Disk-Layout Guidelines" on page 10-30

Once you allocate the necessary space you might still need to take additional steps before **OnLine** 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 performing these tasks:

- Initializing disk space
- Creating a dbspace or blobspace
- Adding a chunk to an existing dbspace or blobspace
- Mirroring an existing dbspace or blobspace

You can allocate 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. See “Creating Links to Each Raw Device” on page 11-6 for more information on this topic.

Allocating Cooked File Space

To allocate cooked file space, you should 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 11-1 illustrates these steps and assumes that you will store the cooked space in the file **/usr/data/my_chunk**.

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).

Figure 11-1 Preparing cooked file space for OnLine

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 10-6), 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.)

Do You Need to Specify an Offset?

You can use offsets for two purposes:

- To prevent **OnLine** 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 the sake of 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 **OnLine** from overwriting critical information required by UNIX. Refer to your disk-drive manuals for the exact offset required.

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** using **onspaces** or **ON-Monitor**. The offset parameter specifies the beginning byte of the chunk. **OnLine** determines the last byte of the chunk by adding the chunk size in bytes to the beginning offset.

For the initial chunk of root dbospace or its mirror, you specify an offset with the `ROOTOFFSET` and `MIRROROFFSET` parameters, respectively. For the initial chunk of nonroot dbospaces, you supply the offset as a parameter when you create the dbospace with `onspaces` or `ON-Monitor`. (See “Creating a Dbospace” on page 11-8.)

If you are running two or more instances of `OnLine`, you must be extremely careful not to define chunks that overlap. Overlapping chunks can cause `OnLine` to overwrite data in one chunk with unrelated data from an overlapping chunk. This 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 quickly replace the disk where the chunk is located. The convenience becomes important if you need to restore your `OnLine` data. The restore process requires that all chunks that were accessible at the time of the last archive are 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 you previously created for the failed device. You do not need to wait for the original device to be repaired.

Do not create file systems on the character-special devices. Do not use the raw device as swap space.

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.

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

Figure 11-2 Listing of /dev directory showing links to raw devices

Figure 11-2 illustrates how a listing appears when a symbolic link is used to link a device to a filename. If your operating system does not support symbolic links, hard links will work as well.

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 **OnLine** database server. This occurs when you bring **OnLine** on-line for the first time.



*Warning: When you initialize **OnLine** disk space, you overwrite whatever is on that disk space. If you reinitialize disk space for an existing **OnLine** database server, all data in the earlier **OnLine** becomes inaccessible and, in effect, is destroyed.*

You execute the **oninit** process by entering the command **oninit** (with or without command-line options) at the UNIX prompt or by requesting initialization through ON-Monitor.

Only user **informix** or **root** can execute **oninit** and initialize **OnLine**. **OnLine** 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 **OnLine** with ON-Monitor, select the Parameters menu, Initialization option. **OnLine** displays a series of five screens, each containing a number of fields that correspond to parameters in the **ONCONFIG** configuration file.

Initializing Disk Space with *oninit*

After you configure **OnLine** (see Chapter 3, “Installing and Configuring **OnLine**”), you can initialize disk space by executing one of the following commands:

```
% oninit -i
```

or

```
% oninit -i -s
```

The **oninit -i** option leaves **OnLine** in on-line mode after initiation. If you use both the **-i** and **-s** options, **OnLine** is left in quiescent mode. Reference information on executing **oninit** is in “Initialize Disk Space and Shared Memory” on page 37-17.

Creating a Dbspace

This section explains how to create a standard dbspace (see “What Is a Dbspace?” on page 10-15) and a temporary dbspace (“What Is a Temporary Dbspace?” on page 10-18). You can use ON-Monitor or **onspaces** for either task.

You can create a dbspace within ON-Monitor or from the command line. Before you do, however, you must first allocate disk space as is described in “Allocating Disk Space” on page 11-3.

You can mirror the dbspace when you create it if mirroring is enabled for **OnLine**. Mirroring takes effect immediately.

Verify that you will not exceed the maximum number of blobspaces and dbspaces allowed in your configuration, specified as DBSPACES. See “DBSPACES” on page 35-13 if you find it necessary to increase the maximum number of dbspaces for your database server.

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 11-6.) 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 11-5.) If you are allocating cooked disk space, the pathname is a file in a UNIX file system.

When the initial chunk of the dbspace you are creating is cooked file space, **OnLine** 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 **OnLine** 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 **OnLine** aware of the existence of the newly created temporary dbspace by setting the DBSPACETEMP configuration variable or the DBSPACETEMP environment variable or both.

You must be logged in as user **informix** or **root** to create a dbspace within ON-Monitor or from the command line.

You can create a dbspace while **OnLine** is in on-line mode. The newly added dbspace (and its associated mirror, if there is one) is available immediately.

Creating a Dbspace Using ON-Monitor

To create a dbspace using 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 a Y in the `Mirror` field. Otherwise, enter N.
4. If the dbspace you are creating is a temporary dbspace, enter a 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 mirror chunk full pathname, size, and optional offset in the mirror chunk section of the screen.

Creating a Dbspace Using *onspaces*

To create a dbspace using **onspaces**, use the `-c` option of **onspaces** as shown in the example that follows.

This example creates a 10 megabyte mirrored dbspace, **dbspce1**. An offset of 5,000 kilobytes is specified for both the primary and mirror chunks.

```
% onspaces -c -d dbspce1 -p /dev/raw_dev1 -o 5000 -s 10000 \  
-m /dev/raw_dev2 5000
```

Reference information on creating a temporary dbspace using **onspaces** is in “Create a Blobspace or Dbspace” on page 37-41.

The following example creates a five megabyte temporary dbspace named **temp_space**.

```
% onspaces -c -t -d temp_space -p /dev/raw_dev1 -o 5000 -s 5000
```

Reference information on creating a dbspace using **onspaces** is in “Create a Blobspace or Dbspace” on page 37-41.

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 11-3.

Adding a Chunk

You add a *chunk* when you need to increase the amount of disk space allocated to a blobospace or dbspace.

Verify that you will not exceed the maximum number of chunks allowed in your configuration, specified as CHUNKS.

If you are adding a chunk to a mirrored blobospace or dbspace, you must also add a mirror 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 11-6 for instructions.

When you add a chunk that is cooked file space, **OnLine** verifies that the disk space is sufficient for the new chunk. If the size of the chunk is greater than the available space on the disk, a message is displayed and no chunk is added. However, the cooked file that **OnLine** created for the new chunk is not removed. Its size represents the space left on your file system before **OnLine** added the chunk. Remove this file to reclaim the space.

When you add a chunk allocated as cooked file space, **OnLine** 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, **OnLine** 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 **OnLine** is in on-line mode. The newly added chunk (and its associated mirror, if there is one) is available immediately.

Adding a Chunk Using ON-Monitor

To add a chunk to a dbspace, follow these instructions:

1. Select the Dbspaces menu, Add_chunk option.
2. Use the RETURN key or the Arrow keys to select the blobospace or dbspace that will receive the new chunk and press CTRL-B or F3.
3. The next screen (see Figure 11-3) that displays indicates whether the blobospace or dbspace is mirrored. If it is, enter a Y in the Mirror field.
4. If the dbspace to which you are adding the chunk is a temporary dbspace, enter a Y in the Temp field.
5. If you indicated that the dbspace or blobospace is mirrored, you must specify both a primary chunk and mirror chunk. Enter the complete pathname for the new primary chunk in the Full Pathname field of the primary chunk section.

```

Press ESC to add new chunk(s).
Press Interrupt to cancel the option and return to the Dbspaces menu.
Press F2 or CTRL-F for field level help.

                ADD CHUNK TO DBSPACE
      Dbspace Name [onchkdbs ]      Mirror [N]      Temp [N]
PRIMARY CHUNK INFORMATION:
      Full Pathname [                ]
      Offset [ 0 ] Kbytes                Size [ 0 ] Kbytes
MIRROR CHUNK INFORMATION
      Full Pathname [                ]
      Offset [  ]

```

Figure 11-3 Add_chunk option of ON-Monitor

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 mirror chunk complete path-name, size, and optional offset in the mirror chunk section of the screen.

Adding a Chunk Using onspaces

To add a chunk to a dbspace, use the **-a** option of **onspaces** as illustrated in the following example. This example adds a 10-megabyte mirrored chunk to **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
```

Reference information on adding a chunk to a dbspace using **onspaces** is in “Add a Chunk” on page 37-42.

Creating a BlobSpace

You can create a blobSpace (see “What Is a BlobSpace?” on page 10-19) using ON-Monitor or the **onspaces** utility. Before you do, however, you must first allocate disk space as is described in “Allocating Disk Space” on page 11-3.

Verify that the number of dbSpaces/blobSpaces does not exceed the current value of the DBSPACES configuration parameter. DBSPACES refers to the total number of blobSpaces plus dbSpaces.

You specify an explicit pathname for the blobSpace. Informix recommends that you use a linked pathname. See “Creating Links to Each Raw Device” on page 11-6 for instructions on how to use a linked pathname.

You can mirror the blobSpace when you create it if mirroring is enabled for **OnLine**. Mirroring takes effect immediately.

A newly created blobSpace is not immediately available for blob storage. BlobSpace logging and recovery require 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 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. (See “Backing Up Logical-Log Files to Free BlobPages” on page 18-16 for instructions.)

When the initial chunk of the blobSpace you are creating is cooked file space, **OnLine** 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 **OnLine** 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 using ON-Monitor or the **onspaces** utility.

You can create a blobSpace while **OnLine** is in on-line mode.

Before you create a blobSpace, take time to determine what blobpage size is optimal for your environment. See “Optimizing BlobSpace Blobpage Size” on page 11-16 for instructions on how to do this.

Determining OnLine Page Size

When you specify blobpage size, you specify it in terms of **OnLine** pages. To determine **OnLine** 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 **OnLine** page size is the last entry on the page. **OnLine** page size is also recorded in the PAGE_PZERO reserved page. You can view the contents of this reserved page by running **oncheck -pr**.

You can also find **OnLine** page size by selecting the Initialize option of the parameters menu. The first entry displayed on the screen after you select the Initialize option is **OnLine** page size.

Creating a BlobSpace Using ON-Monitor

To create a blobspace using 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 blobpage chunk, enter a Y in the Mirror field. Otherwise, enter N.
4. Specify the blobpage size in terms of the number of disk pages (see "Determining OnLine Page Size" on page 11-13) per blobpage in the BLOBPage Size field. For example, if your **OnLine** has a disk-page size of 2 kilobytes, and you want your blobpages to have a size of 10 kilobytes, enter a 5 in this field.
5. Enter the complete pathname for the initial primary chunk of the blob-space 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 Using *onspaces*

To create a blobspace using **onspaces**, use the **-c** option as illustrated in the following example. This example creates a ten megabyte mirrored blobspace, **blobsp3**, with a blobpage size of 10 kilobytes, where **OnLine** page size is 2 kilobytes. An offset of 200 kilobytes for the primary and mirror chunks is specified.

```
% onspaces -c -b blobsp3 -g 5 -p /dev/raw_dev1 -o 200 -s 10000 \  
-m /dev/raw_dev2 200
```

Reference information on creating a blobspace using **onspaces** is in “Create a Blobspace or Dbspace” on page 37-41.

Adding a Chunk to a Blobspace

Adding a chunk to a blobspace is identical to adding a chunk to a dbspace. Both are explained in “Adding a Chunk” on page 11-10.

Dropping a Chunk from a Dbspace Using *onspaces*

To successfully drop a chunk from a dbspace using **onspaces**, all pages other than overhead pages must be freed. If any pages remain allocated to non-overhead entities, **onspaces** returns the following error:

```
Chunk is not empty.
```

If this occurs, execute **oncheck -pe** to determine which **OnLine** entity still occupies space in the chunk, remove it, and reenter the **onspaces** command.

You cannot drop the initial chunk of a dbspace. (Use the **fchunk** column of **onstat -d** to determine which chunk is the initial chunk of a dbspace. (See “-d Option” on page 37-52 for more information.)

The following example drops a chunk from **dbsp3**. An offset of 300 kilobytes is specified.

```
% onspaces -d dbsp3 -p /dev/raw_dev1 -o 300
```

Reference information on dropping a chunk from a dbspace using **onspaces** is in “Drop a Chunk” on page 37-43.

Dropping a Chunk from a BlobSpace

The procedure for dropping a chunk from a blobSpace is identical to the procedure for dropping a chunk from a dbSpace described in “Dropping a Chunk from a DbSpace Using onSpaces” on page 11-14 except that **OnLine** must be in quiescent mode. Other than this, you need only substitute the name of your blobSpace wherever there is a reference to a dbSpace.

Dropping a DbSpace or BlobSpace

Before you drop a dbSpace, you must first drop all databases and tables that you previously created in the dbSpace. Before you drop a blobSpace, you must drop all tables that have a TEXT or BYTE column referencing the blobSpace.

Execute **oncheck -pe** to verify that no tables or log files are residing in the dbSpace or blobSpace.

You cannot drop the root dbSpace.

After you drop a dbSpace or blobSpace, the newly freed chunks are available for reassignment to other dbSpaces or blobSpaces. However, before you reassign the newly freed chunks, you should perform a level-0 archive. If you are using ON-Archive, the level-0 archive, should include at least the root dbSpace and the dbSpace set (if any) that contained the dropped dbSpace or blobSpace.

If you drop a dbSpace or blobSpace that is mirrored, the dbSpace or blobSpace mirrors are also dropped.

If you want to drop only the dbSpace or blobSpace mirrors, turn off mirroring. (See “Ending Mirroring” on page 24-10.) This drops the dbSpace or blobSpace mirrors and frees the chunks for other uses.

You must be logged in as **root** or **informix** to drop a dbSpace from either ON-Monitor or **onSpaces**.

You can drop a dbSpace while **OnLine** is in on-line mode.

Dropping a Dbspace or BlobSpace Using ON-Monitor

To drop a dbspace or blobSpace using ON-Monitor, follow these instructions:

1. Select the Dbspaces menu, Drop option.
2. Use the RETURN key or Arrow keys to scroll to the dbspace or blobSpace 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 or BlobSpaces Using *onspaces*

To drop a dbspace or blobSpace using *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
```

Reference information on dropping a dbspace or blobSpace using *onspaces* is in “Drop a BlobSpace or Dbspace” on page 37-42.

Optimizing BlobSpace Blobpage Size

It is helpful to familiarize yourself with the **OnLine** approach to blobSpace blob storage before you begin this section. “Structure of a BlobSpace” on page 40-54 and “BlobSpace Page Types” on page 40-59 provide background information for this section. This section is not applicable if you store blobs in tables.

Determining BlobSpace Storage Efficiency

When you are evaluating blobSpace storage strategy, you can measure efficiency by two criteria:

- Blobpage fullness
- Blobpages required per blob

Blobpage fullness refers to the amount of data within each blobpage. Blobs stored in a blobSpace cannot share blobpages. Therefore, if a single blob 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 blob, you can increase the overhead cost of storage. For example, more locks are required for updates since 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 two **OnLine** 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 blob stored as part of the table (or database)

The **oncheck -pe** command can provide background information about the blobs 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 **OnLine** pages used by each table to store its associated blob 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 blobs in a database or table. If you find that the statistics for a significant number of blobs show a low percentage of fullness, **OnLine** might benefit from resizing the blobpage in the blobSpace.

The following example retrieves storage information for all blobs stored in the table **sriram.catalog** in the **stores6** database.

```
% oncheck -pB stores6:sriram.catalog
```

Figure 11-4 shows the output of this command.

```

BLOBSpace Report for stores6:sriram.catalog
Total pages used by table          7
BLOBSpace usage:
Space Name      Page Number  Pages  0-25%  Percent Full  51-75%  76-100%
-----
blobPIC        0x300080    1                x
blobPIC        0x300082    2                x
      Page Size is 6144    3
bspcl          0x2000b2    2                                x
bspcl          0x2000b6    2                                x
      Page Size is 2048    4
    
```

Figure 11-4 Output of oncheck -pB

Space Name is the name of the blobpage that contains one or more blobs stored as part of the table (or database).

Page Number is the starting address in the blobpage of a specific blob.

Pages is the number of **OnLine** pages required to store this blob.

Percent Full is a measure of the average fullness of all the blobpages that hold this blob.

Page Size is the size in bytes of the blobpage for this blobpage. Blobpage size is always a multiple of the **OnLine** page size. (See “Determining OnLine Page Size” on page 11-13 for instructions on how to obtain the page size for your database server.)

The example output indicates that four blobs are stored as part of the table **sriram.catalog**. Two blobs are stored in the blobpage **blobPIC** in 6144-byte blobpages. Two more blobs are stored in the blobpage **bspcl** 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 blobs. The total says nothing about the size of the blobpages used, the number of blobs 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 blob storage. To understand how the fullness statistics can improve your blob storage strategy, it is helpful to use the example output in Figure 11-4 on page 11-18 to explain the idea of average fullness.

Interpreting Blobpage Average Fullness

The first blob listed in Figure 11-4 is stored in the blobspace **blobPIC** and requires one 6144-byte blobpage. The blobpage is 51 to 75 percent full, meaning that the minimum blob size is between $0.51 * 6104 = 3072$ bytes and $0.75 * 6144 = 4508$. The maximum size of this blob must be less than or equal to 75 percent of 6144 bytes, or 4508 bytes.

The second blob 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 blob must be greater than 50 percent of 12,288 bytes, or 6144 bytes. The maximum size of the blob 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 blobs in blobspace **bsp1**. These two blobs appear to be nearly the same size. Both blobs require two 2048-byte blobpages and the average fullness for each is 76 to 100 percent. The minimum size for these blobs must be greater than 75 percent of the allocated blobpages, or 3072 bytes. The maximum size for each blob is slightly less than 4096 bytes (allowing for overhead).

Apply Efficiency Criteria to Output

Looking at the efficiency information for blobspace **bsp1**, an **OnLine** administrator might decide that a better blob-storage strategy would be to double the blobpage size from 2048 bytes to 4096 bytes. (Recall that blobpage size is always a multiple of the **OnLine** page size.) If the **OnLine** administrator made this change, the measure of page fullness would remain the same but the number of locks needed during a blob update or modification would be reduced by half.

The efficiency information for blobspace **blobPIC** reveals no obvious suggestion for improvement. The two blobs in **blobPIC** differ considerably in size and there is no optimal storage strategy. In general, blobs of similar size can be stored more efficiently than blobs of different sizes.

Managing Extents

You should periodically monitor **OnLine** chunks to check for extent interleaving. (See “What Is Extent Interleaving?” on page 10-26.)

Execute **oncheck -pe** to obtain the physical layout of information in the chunk. The chunk layout is sequential, and the number of pages dedicated to each table is shown. The following information displays:

- Dbspace name, owner, and number
- Number of chunks in the dbspace

This output is useful for determining the amount of extent interleaving. If **OnLine** is unable to allocate an extent in a chunk despite an adequate number of free pages, the chunk might be badly fragmented.

If You Find That Extents Are Interleaved

Depending on the specific circumstances, you might be able to eliminate interleaving by using the **TO CLUSTER** option of **ALTER INDEX** statement to rebuild the tables. The **TO CLUSTER** option causes the reordering of rows in the physical table to the indexed order.

For this tactic to work, the chunk must contain adequate contiguous space in which to rebuild each table. In addition, the contiguous space in the chunk must be the space that **OnLine** normally allocates to rebuild the table. (That is, **OnLine** allocates space for the **ALTER INDEX** processing from the beginning of the chunk, looking for blocks of free space that are greater than or equal to the size specified for the **NEXT EXTENT**. If the contiguous space is located near the end of the chunk, **OnLine** could rebuild the table using blocks of space that are scattered throughout the chunk.)

Use the **TO CLUSTER** option of the **ALTER INDEX** statement on every table in the chunk. Follow these steps:

1. For each table, drop all the indexes except one.
2. Cluster the remaining index using the **TO CLUSTER** option of the **ALTER INDEX** statement.
3. Re-create all the other indexes.

You eliminate the interleaving in the second step, when you rebuild the table by rearranging the rows. In the third step, you compact the indexes as well because the index values are sorted before they are added to the B+ tree. You do not need to drop an index before you cluster it. However, if you do, the **ALTER INDEX** processing is faster and you gain the benefit of more compact indexes.

A second solution to extent interleaving is to unload and reload the tables in the chunk.

To prevent the problem from recurring, consider increasing the size of the tblspace extents. See Chapter 10 of the *Informix Guide to SQL: Tutorial* for more information.

Managing Tables

Once disk space has been allocated to a tblspace as part of an extent, that space remains dedicated to the tblspace. Even if all extent pages become empty as a result of deleting data, the disk space remains unavailable for use by other tables.

As **OnLine** administrator, you can reclaim the disk space in empty extents and make it available to other users by rebuilding the table. You can accomplish this in the following ways:

- Use ALTER INDEX
- Use LOAD and UNLOAD

Each of these methods is described in the following sections.

Reclaiming Space in an Empty Extent Using Alter Index

If the table with the empty extents includes an index, you can execute the ALTER INDEX statement with the TOCLUSTER keywords. Clustering an index rebuilds the table in a different location within the dbspace. All the extents associated with the previous version of the table are released. Also, the newly built version of the table has no empty extents.

Reclaiming Space in an Empty Extent Using the UNLOAD and LOAD Statements

If the table does not include an index, you can unload the table, re-create the table (either in the same dbspace or in another), and reload the data using **OnLine** utilities or the UNLOAD and LOAD statements. For further information about selecting the correct utility or statement to use, refer to “Summary of Methods for Moving Data” on page 31-4.

What Is the Dynamic Scalable Architecture?

Chapter Overview	3
What Is a Virtual Processor?	4
What Is a Thread?	5
What Is a User Thread?	5
Types of Virtual Processors	5
Advantages of Virtual Processors	6
Virtual Processors Can Share Processing	7
Virtual Processors Save Memory and Resources	7
Virtual Processors Can Do Parallel Processing	8
You Can Add and Drop Virtual Processors While OnLine Is in On-Line Mode	9
You Can Bind Virtual Processors to CPUs	9
How Virtual Processors Service Threads	9
Control Structures	10
Context Switching	10
Stacks	12
Queues	14
Ready Queues	14
Sleep Queues	14
Wait Queues	15
Mutexes	16

Virtual Processor Classes	16
CPU Virtual Processors	16
How Many CPU Virtual Processors Do You Need?	16
Running on a Multiprocessor Computer	17
Running on a Single-Processor Computer	17
Adding and Dropping CPU Virtual Processors While OnLine Is On-Line	18
Preventing Priority Aging	18
Using Processor Affinity	18
Disk I/O Virtual Processors	20
I/O Priorities	21
Logical-Log I/O	21
Physical-Log I/O	22
Asynchronous I/O	22
Network Virtual Processors	24
Specifying Network Protocols	25
Should Poll Threads Run on CPU or Network Virtual Processors?	25
How Many Networking Virtual Processors Do You Need?	25
Listen and Poll Threads—How the Client/Server Connection Works	26
Starting Multiple Listen Threads	29
Administration Virtual Processors	30
Optical Virtual Processor	30
Audit Virtual Processor	31

Chapter Overview

The **INFORMIX-OnLine Dynamic Server** implements an advanced RDBMS architecture that Informix calls the *Dynamic Scalable Architecture* (DSA). DSA provides distinct performance advantages over previous versions of **INFORMIX-OnLine** for both single-processor and multiprocessor platforms. These advantages, which this chapter describes further, are as follows:

- A small number of **OnLine** 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

The **INFORMIX-OnLine Dynamic Server** particularly exploits *symmetric multiprocessing* computer systems (SMPs). A symmetric multiprocessing computer system is one in which 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 **OnLine** provides on these systems:

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

The central component of the dynamic scalable architecture is the *virtual processor*, which is described in the following section. See Chapter 13, “Managing Virtual Processors,” for information on how to configure **OnLine** virtual processors.

What Is a Virtual Processor?

The **INFORMIX-OnLine** database server processes are called *virtual processors*. They 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, an **OnLine** virtual processor runs multiple *threads* to service multiple SQL client applications.

Figure 12-1 illustrates the relationship of client applications to virtual processors in the dynamic server.

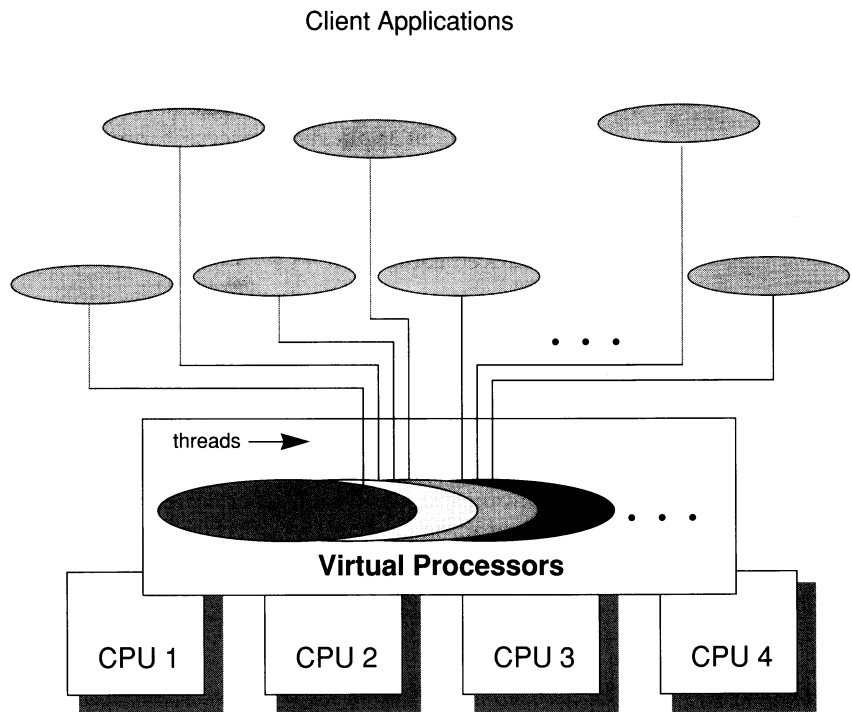


Figure 12-1 *In the dynamic scalable architecture, a small number of virtual processors serves a much larger number of client applications.*

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.

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

An **OnLine** virtual processor runs threads on behalf of SQL client applications (*session threads*) and to satisfy internal requirements (*internal threads*). In most cases, for each connection by a client application, **OnLine** runs one session thread. **OnLine** runs internal threads to accomplish, among other things, database I/O, logging I/O, page cleaning, and administrative tasks. See “Virtual Processors Can Do Parallel Processing” on page 12-8 for cases in which **OnLine** runs multiple session threads for a single client.

What Is a User Thread?

A *user thread* is an **OnLine** thread that services requests from client applications. User threads include session threads, called **sqlexec** threads, which are the primary threads that **OnLine** 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.

Types of Virtual Processors

Virtual processors are divided into *classes* based on the type of processing that they do. Each class of virtual processor is dedicated to processing certain types of threads. Figure 12-2 shows the classes of virtual processors and the types of processing that they do:

Virtual Processor Class	Purpose
CPU	Runs all session threads and some system threads. Runs thread for kernel asynchronous I/O where available.
	Disk I/O
PIO	Writes to the physical-log file (internal class) if it is in cooked disk space.
LIO	Writes to the logical-log files (internal class) if they are in cooked disk space.
AIO	Performs nonlogging disk I/O. If kernel asynchronous I/O is used, AIO virtual processors perform I/O to cooked disk spaces.
	Network
SHM	Performs shared memory communication.
TLI	Performs network communication using TLI.
SOC	Performs network communication using sockets.
	Optical
OPT	Performs I/O to optical disk.
	Administrative
ADM	Performs administrative functions.
	Auditing
ADT	Performs auditing functions.

Figure 12-2 Virtual processor classes

Advantages of Virtual Processors

When compared to a database server process that services a single client application, the dynamic, multithreaded nature of an **OnLine** 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 **OnLine** is running.
- You can bind virtual processors to CPUs.

This section describes these advantages.

Virtual Processors Can Share Processing

Virtual processors in the same class have identical code and share access to both data and processing queues in memory. This means that any virtual processor in a class can run any thread that belongs to that class.

Generally, **OnLine** 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 processor's memory to be transferred on the bus. When a thread is waiting to run, however, **OnLine** migrates the thread to another virtual processor because the benefit of balancing the processing load outweighs the amount of overhead incurred in transferring the data.

Shared processing within a class of virtual processors occurs automatically and is transparent to the database user.

Virtual Processors Save Memory and Resources

OnLine is able to service a large number of clients with a small number of server processes when 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 it.

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 one. This means that both processes must enter and exit the operating system kernel and that 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. See "Context Switching" on page 12-10 for a description of how a virtual processor switches from one thread to another.

Virtual Processors Can Do Parallel Processing

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

Figure 12-3 illustrates parallel processing. When a client initiates index building, sorting, or recovery, **OnLine** spawns multiple threads to work on the task in parallel, using as much of the computer's resources as possible. While one thread is waiting on I/O, another can be working.

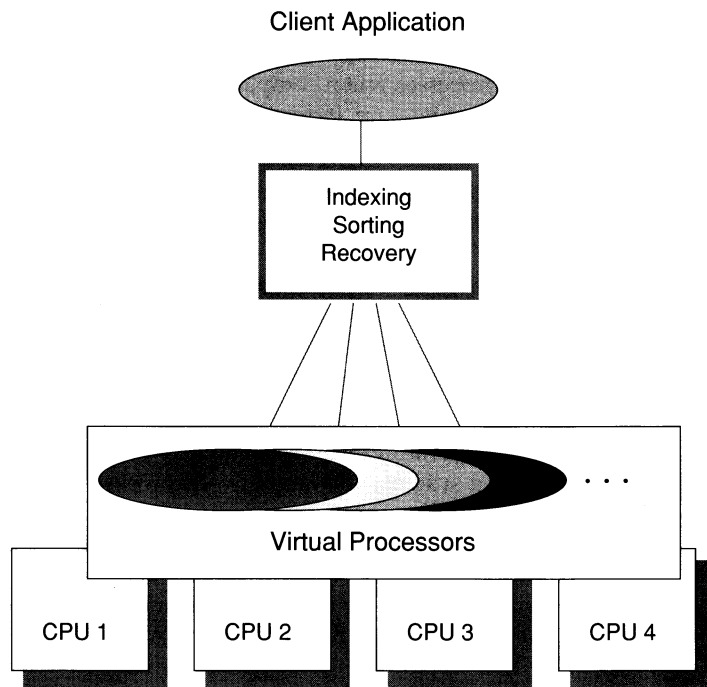


Figure 12-3 When building indexes, sorting, or performing recovery **OnLine** spawns multiple threads to work in parallel for a single client, using as much of the computer's resources as possible.

You Can Add and Drop Virtual Processors While OnLine Is in On-Line Mode

You can add virtual processors to meet increasing demands for service while **OnLine** is running. For example, if the virtual processors of a class become compute 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 further distribute the processing load.

While **OnLine** is running, you can add virtual processors for any of the classes but you can only drop virtual processors for the CPU class. See “Adding Virtual Processors in On-Line Mode” on page 13-7 and “Dropping CPU Virtual Processors in On-Line Mode” on page 13-9 for information on how to add or drop virtual processors while **OnLine** is in on-line mode.

You Can Bind 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 machines for which **OnLine** supports processor affinity, you can bind CPU virtual processors to specific CPUs in the computer. When you bind a CPU virtual processor to a CPU, the virtual processor runs exclusively on that CPU. This improves the performance of the CPU virtual processor because it reduces the amount of switching between processes that the operating system must do. Binding CPU virtual processors to specific CPUs also enables you to isolate database work to specific processors on the computer, leaving the remaining processors free for other work.

See “Using Processor Affinity” on page 12-18 for information on how to assign CPU virtual processors to hardware processors.

How Virtual Processors Service Threads

At a given moment in time, a virtual processor can run only one thread. A virtual processor services multiple threads concurrently by switching between them—that is, by running a thread until it yields and then switching to another one, and, likewise, to another one, eventually returning to the original thread when it is ready to continue. All the while, some threads complete their work and the virtual processor starts new threads to complete new work. By continually switching between threads, the virtual processor is able to 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. **OnLine** 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 **OnLine**, **OnLine** creates a *session* structure, 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, **OnLine** creates a thread structure, called a *thread-control block* (tcb) for the session and initiates a primary thread (**sqlxec**) 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, **OnLine** 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*. **OnLine** does not preempt a running thread, as the operating system does to a process, when a fixed amount of time (time-slice) expires. Rather, 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. Thus, when a thread performs one of these long-running tasks, it will yield 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 **OnLine** to interrupt threads at points that are most advantageous from a performance standpoint.

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 a 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 12-4 illustrates how a virtual processor accomplishes a context switch.

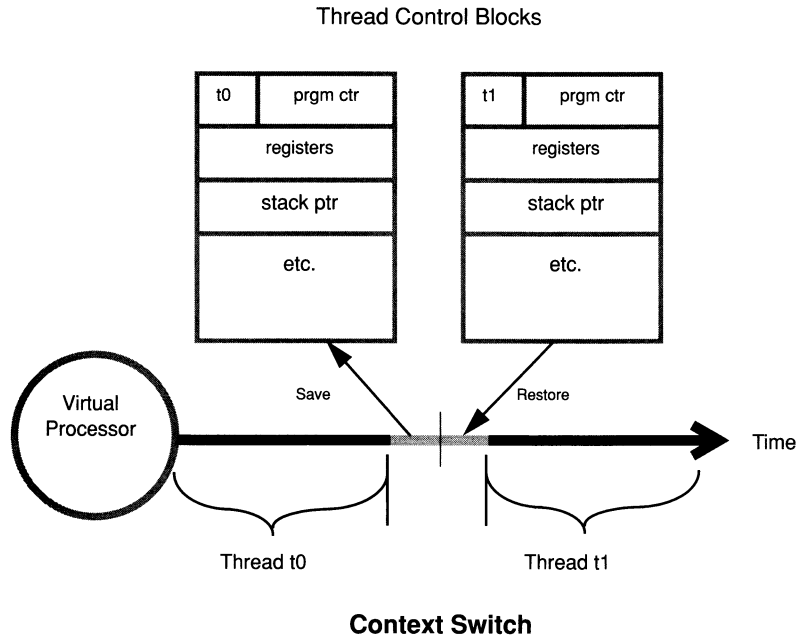


Figure 12-4 Context switch: how a virtual processor switches from one thread to another

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 is called the thread's *stack*. See "Stacks" on page 14-28 for information on how to set the size of the stack.

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 belonging 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 off-sets to the beginning address to access data within the stack. Figure 12-5 illustrates how a virtual processor uses the stack to segregate nonshared data for session threads.

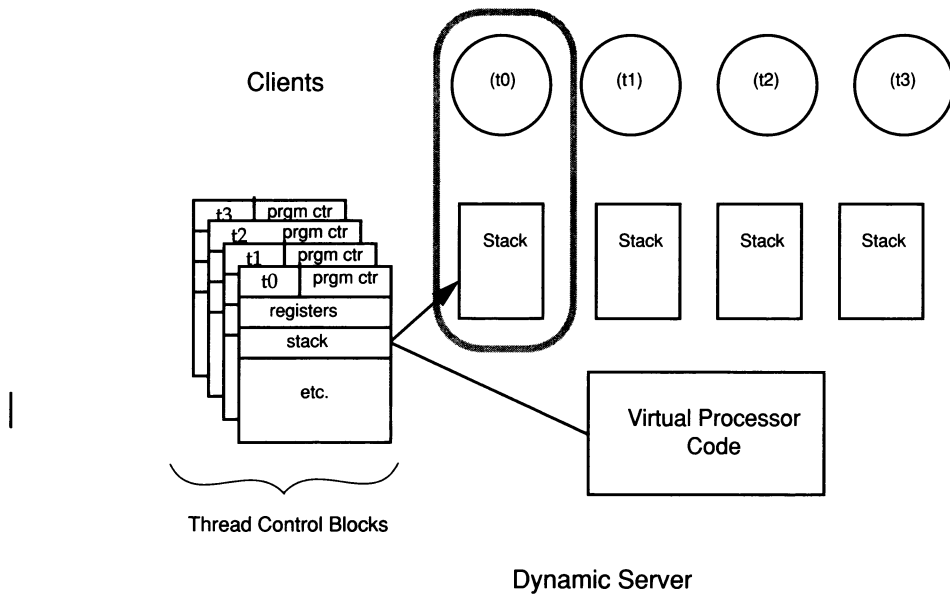


Figure 12-5 Virtual processors segregate nonshared data for each user

Queues

The dynamic server uses three types of queues to schedule the processing of multiple, concurrently running threads:

- Ready queues
- Wait queues
- Sleep queues

Virtual processors of the same class share queues. This fact, in part, enables a thread to migrate, when necessary, from one virtual processor in a class to another.

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 having 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. See “Monitoring Virtual Processors” on page 29-27 for information on how to monitor the ready queues. See “Adding Virtual Processors in On-Line Mode” on page 13-7 for information on how to add virtual processors while **OnLine** is in on-line mode.

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 for *forever*.

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 12-6 illustrates how **OnLine** threads are queued to perform database I/O.

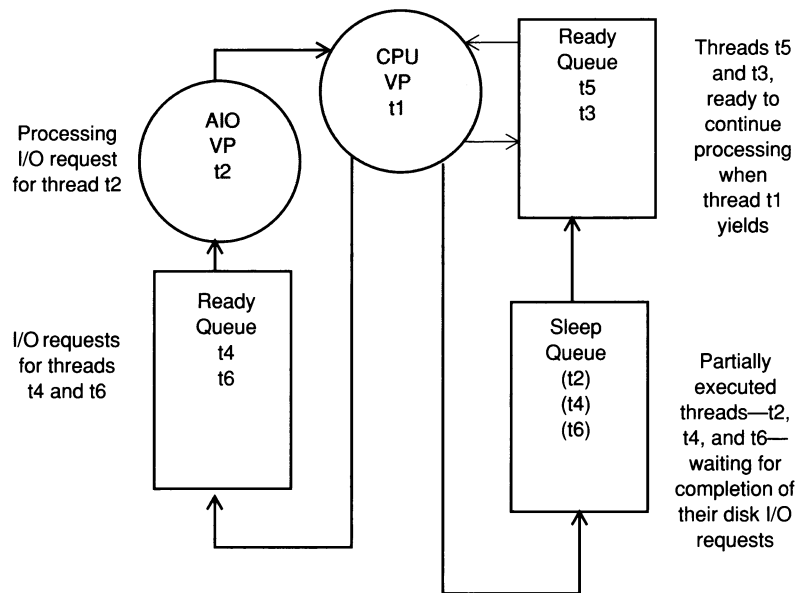


Figure 12-6 How OnLine threads are queued to perform database I/O

Wait Queues

Wait queues hold threads that need to wait for a particular event before they can continue to run. Wait queues, for example, coordinate access to shared data by threads. When a user thread tries to acquire the logical-log latch and 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 to see if there are threads waiting and, if so, it wakes up the next thread in the wait queue.

Mutexes

A mutex (*mutually exclusive*) is a latching mechanism that **OnLine** 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*. Mutexes permit a greater degree of parallelism, however, 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 it 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.

See “Monitoring Latches” on page 29-22 for information on monitoring mutexes (latches).

Virtual Processor Classes

A virtual processor of a given class can only run threads of that class. This section describes the types of threads, or the types of processing, done by each class of virtual processor. 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 **OnLine**. For example, a thread that listens for connection requests from client applications is an internal thread.

How Many CPU Virtual Processors Do You Need?

The right number of CPU virtual processors is the number at which they are all kept busy but not so busy that they cannot keep pace with incoming requests. You should not allocate more CPU virtual processors than the number of hardware processors in the computer.

The NUMCPUVPS parameter in the ONCONFIG file specifies the number of CPU virtual processors that **OnLine** brings up initially. See “Setting Virtual Processor Configuration Parameters” on page 13-3 for information on setting

the NUMCPUVPS parameter. See “Should Poll Threads Run on CPU or Network Virtual Processors?” on page 12-25 for an additional consideration in deciding how many CPU virtual processors you need.

On some two-processor computers, you should run only one CPU virtual processor because the overhead of synchronizing the two virtual processors offsets the benefit of distributing the processing.

For **OnLine** platforms with more than two CPUs, if the computer will be primarily a database server, Informix recommends that you begin by setting the number of virtual processors to one less than the number of CPUs in the computer. If the computer will not be used primarily as a database server, Informix recommends that you start by specifying one CPU virtual processor. Then increase or decrease the number of CPU virtual processors as indicated by performance.

To evaluate the performance of the CPU virtual processors while **OnLine** 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, **OnLine** performs locking in a manner that is appropriate for a multiprocessor computer. See “MULTIPROCESSOR” on page 35-27 for information on setting multiprocessor mode.

Running on a Single-Processor Computer

If you are running only one CPU virtual processor, set the SINGLE_CPU_VP configuration parameter to 1 and the MULTIPROCESSOR configuration parameter to 0. Setting the SINGLE_CPU_VP parameter to 1 allows **OnLine** to bypass some of the mutex calls that **OnLine** normally makes when it runs multiple CPU virtual processors. Setting MULTIPROCESSOR to 0 enables **OnLine** to bypass the locking that is required for multiple processes on a multiprocessor computer. See “SINGLE_CPU_VP” on page 35-40 for information on setting the SINGLE_CPU_VP parameter.

Note: 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. You must set `SINGLE_CPU_VP` to 1 to reduce the amount of latching that will be done when you run a single CPU virtual processor.

If you set the `SINGLE_CPU_VP` parameter to 1, the value of the `NUMCPUVPS` parameter must also be 1; if the latter is greater than 1, **OnLine** fails to initialize and displays the message:

```
Cannot have 'SINGLE_CPU_VP' non-zero and 'NUMCPUVPS' greater than 1
```

If the `SINGLE_CPU_VP` parameter is set to 1, you are not able to add CPU virtual processors while **OnLine** is in on-line mode. See “Adding Virtual Processors in On-Line Mode” on page 13-7 for more information.

Adding and Dropping CPU Virtual Processors While OnLine Is On-Line

You can add or drop CPU class virtual processors while **OnLine** is on-line. For instructions on how to do this, see “Adding Virtual Processors in On-Line Mode” on page 13-7 and “Dropping CPU Virtual Processors in On-Line Mode” on page 13-9.

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.

If your operating system allows you to disable priority aging, you can disable it for **OnLine** virtual processors by setting the `NOAGE` parameter in the `ONCONFIG` file. See “Setting Virtual Processor Configuration Parameters” on page 13-3 for information on how to set this parameter.

Using Processor Affinity

On some multiprocessor platforms that support *processor affinity*, you can assign CPU virtual processors to specific CPUs. When you assign a CPU virtual processor to a specific CPU, the virtual processor runs exclusively on that CPU. See the **OnLine** machine notes file, which is described under “Useful On-Line Files” in the Introduction to see if processor affinity is supported on your **OnLine** platform.

You must 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. You should not set AFF_NPROCS to a number that is less than the number of CPU virtual processors you have allocated—that is, the number of CPUs should not be less than the number of CPU virtual processors that you allocate.

Set the AFF_SPROC parameter to the number of the first CPU to which a CPU virtual processor should be assigned. **OnLine** assigns CPU virtual processors to CPUs in serial fashion, starting with this processor. The first processor is number 0. For example, if you have four CPUs (AFF_NPROCS = 3), and you set NUMCPUVPS to 3 and AFF_SPROC to 1, the three CPU virtual processors are assigned to the second, third and fourth CPUs, respectively. If you set AFF_SPROC to 2, **OnLine** would display an error message, indicating that the computer does not have enough CPUs to use affinity. The value of AFF_NPROCS plus the value of AFF_SPROC must be less or equal to the number of physical processors. In this case, 3 (AFF_NPROCS) plus 1 (AFF_SPROC) equals four, the number of physical processors. Figure 12-7 illustrates how **OnLine** uses the AFF_NPROCS and AFF_SPROC parameters to implement processor affinity.

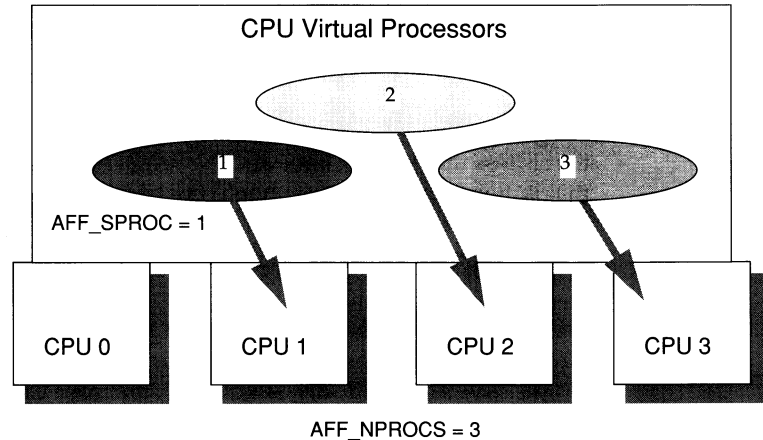


Figure 12-7 When using processor affinity, `AFF_NPROCS` and `AFF_SPROC` determine how CPU virtual processors are assigned to CPUs.

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)

OnLine 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. **OnLine** uses the CPU class to perform *kernel asynchronous I/O* (KAIO) when it is available on a platform. If **OnLine** 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. See "Asynchronous I/O" on page 12-22 for more information about nonlogging I/O.

The PIO class performs all I/O to the physical-log file and the LIO class performs all I/O to the logical-log files, *unless* they reside in raw disk space and **OnLine** has implemented kernel asynchronous I/O.

I/O Priorities

In general, **OnLine** 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. This 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. **OnLine** prioritizes the different types of disk I/O that it performs as shown in Figure 12-8.

Priority	Type of I/O	VP Class
Highest	Logical-log I/O	CPU or LIO
	Physical-log I/O	CPU or PIO
	Database I/O	CPU or AIO
	Page-cleaning I/O	CPU or AIO
Lowest	Read-ahead I/O	CPU or AIO

Figure 12-8 *How OnLine prioritizes disk I/O*

Logical-Log I/O

If kernel asynchronous I/O is implemented and the logical-log files are in raw disk space, **OnLine** uses a KAIO thread in the CPU virtual processor to perform I/O to the logical log. If kernel asynchronous I/O is not implemented, or the logical-log files are in cooked disk space, the LIO class of virtual processors performs I/O to the logical-log files.

The logical-log files store the data that enables **OnLine** to roll back transactions and recover from system failures. I/O to the logical-log files is the highest priority disk I/O that **OnLine** performs.

If the logical-log files are in a dbspace that *is not* mirrored, **OnLine** runs only one LIO virtual processor. If the logical-log files are in a dbspace that *is* mirrored, **OnLine** runs two LIO virtual processors. This class of virtual processors has no parameters associated with it.

Physical-Log I/O

If kernel asynchronous I/O is implemented and the physical-log file is in raw disk space, **OnLine** uses a KAIO thread in the CPU virtual processor to perform I/O to the physical log. If kernel asynchronous I/O is not implemented, or the physical-log file is in cooked disk space, the PIO class of virtual processors performs I/O to the physical-log file.

The physical-log file stores *before images* of dbspace pages that have changed since the last *checkpoint*. (See “OnLine Checkpoints” on page 14-47 for more information on checkpoints.) At the start of recovery, prior to processing transactions from the logical log, **OnLine** 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, **OnLine** runs only one PIO virtual processor. If the physical-log file is in a dbspace that *is* mirrored, **OnLine** runs two PIO virtual processors. This class of virtual processors has no parameters associated with it.

Asynchronous I/O

OnLine performs database I/O asynchronously, meaning that I/O is queued and later performed independent of the process requesting the I/O. Performing I/O asynchronously allows the process that makes the request to continue working while the I/O is being done.

OnLine 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 the AIO class of 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

OnLine 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.

OnLine implements kernel asynchronous I/O by running a KAIO thread in the CPU virtual processor. The KAIO thread performs an I/O by making system calls to the operating system, which performs the I/O independent of the

virtual processor. The KAIO thread can produce better performance for disk I/O than the AIO virtual processor because it does not require a switch between the CPU and AIO virtual processors.

Informix implements kernel asynchronous I/O when it ports **OnLine** to a platform that supports it; it is not implemented by the **OnLine** administrator. See the **OnLine** machine notes file, which is described under “Useful On-Line Files” in the Introduction, to see if kernel asynchronous I/O is supported on your computer.

AIO Virtual Processors

If the platform does not support kernel asynchronous I/O, or if the I/O is to cooked disk space, **OnLine** performs database I/O through the AIO class of virtual processors.

As each AIO virtual processor comes up, **OnLine** assigns it to a *home queue*. If all disks have requests, the AIO virtual processors should be spread evenly across the queues to keep all of the disks busy. If only a subset of the disks have requests, those AIO virtual processors that do not have requests begin processing requests from those queues that do.

You use the NUMAIOVPS parameter in the ONCONFIG file to specify the number of AIO virtual processors that **OnLine** brings up initially. See “Setting Virtual Processor Configuration Parameters” on page 13-3 for information on how to set this parameter.

You can start additional AIO virtual processors while **OnLine** is in on-line mode. See “Adding Virtual Processors in On-Line Mode” on page 13-7 for information on how to do this.

You cannot drop AIO virtual processors while **OnLine** 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/Os to the disk devices are being processed as fast as they occur. The **onstat -g ioq** command allows you to monitor the length of the I/O queues for the AIO virtual processors. See “Monitoring Virtual Processors” on page 29-27 for more information.

If **OnLine** 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 **OnLine** 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 per each disk that **OnLine** accesses frequently.

You should 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, “Configuring Connectivity,” a client can connect to **OnLine** 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*). See “Using a Local Loopback Connection” on page 4-21.

A client can connect to **OnLine** through four types of connections (ipcshm, socketp, tlitcp, and tlisp). A network virtual processor supports each connection type as shown in Figure 12-9:

Network VP	Connection Type
SHM	Shared memory (local only)
SOC	TCP/IP using Sockets (local or remote)
TLI	TCP/IP using Transport Level Interface (local or remote)
	<i>or</i>
	Sequenced Packet Exchange (SPX) using Transport Level Interface (Portable NetWare)

Figure 12-9 *OnLine network virtual processor*

On a UNIX computer, **OnLine** supports either the *sockets* or the *TLI* mechanism to interface to networking facilities, depending on which mechanism the platform supports. To determine which mechanism **OnLine** supports on your platform, see the **OnLine** machine notes file, which is described under “Useful On-Line Files” in the Introduction. The IPX/SPX protocol, which uses TLI, enables you to connect to a Novell Netware machine.

Specifying Network Protocols

The DBSERVERNAME, DBSERVERALIASES, and NETTYPE parameters in the ONCONFIG file determine which protocols **OnLine** will use.

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 a network protocol. **OnLine** runs one or more *poll threads* for each unique **nettype** entry defined by a dbservername in the **sqlhosts** file. See “The nettype Field” on page 4-12 for a description of the **nettype** field.

The NETTYPE configuration parameter provides optional configuration information for a protocol. It allows you to allocate more than one poll thread for a protocol and also designate the virtual processor class (CPU or NET) that will run the poll threads. See “NETTYPE” on page 35-28 for a complete description of this parameter.

Should Poll Threads Run on CPU or Network Virtual Processors?

Poll threads can run either *inline* on CPU virtual processors or they run on network virtual processors for the particular protocol (SHM, SOC, TLI). In general, and particularly on a single-processor computer, poll threads run more efficiently on CPU virtual processors. This may 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 protocol (poll threads) associated with the DBSERVERNAME variable, the class defaults to CPU. **OnLine** assumes that the protocol associated with DBSERVERNAME is the primary protocol and that it should be the most efficient.

For other protocols, if no `vp class` is specified, the default is NET.

While **OnLine** 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 a protocol 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 **OnLine** 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 protocol 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 protocol.

Listen and Poll Threads—How the Client/Server Connection Works

When you start **OnLine**, the **oninit** process starts an internal thread, called a *listen thread*, for each **dbservername** that you specify with the **DBSERVER-NAME** and **DBSERVERALIAS** 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 12-15 would cause the **OnLine** database server **soc_ol1** to start a listen thread for **port1** on the host, or network address, **myhost**.

dbservername	protocol	hostname	service name
soc_ol1	onsoctcp	myhost	port1

Figure 12-10 OnLine starts a listen thread for each listen port that you specify in the sqlhosts file

The listen thread opens the port and requests one of the poll threads for the specified protocol 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 protocol that is being used (SHM, SOC, or TLI). See “Specifying Network Protocols” on page 12-25 for information on specifying how many poll threads **OnLine** runs. See “Should Poll Threads Run on CPU or Network Virtual Processors?” on page 12-25, and “NETTYPE” on page 35-28, for information on how to specify whether the poll threads for a protocol run in CPU or network virtual processors.

When a poll thread receives a connection request from a client, it passes it to the listen thread for the port. The listen thread authenticates the user, establishes the connection to **OnLine**, and starts an **sqlxec** thread, the session thread that does the primary processing for the client. Figure 12-11 illustrates the roles of the listen and poll threads in establishing a connection with a client application.

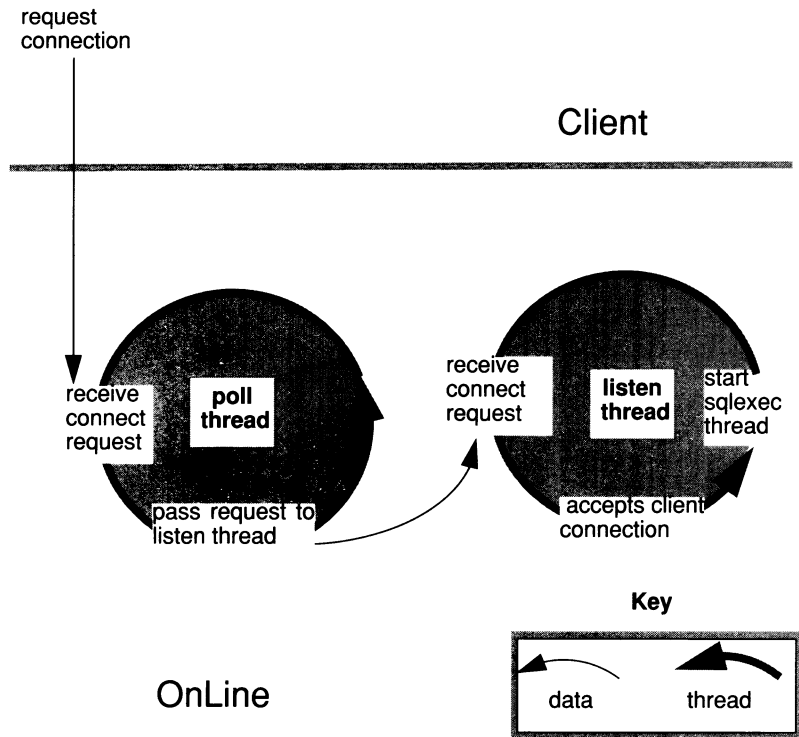


Figure 12-11 *The roles of the poll and the listen threads in connecting to a client*

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 12-12 illustrates the basic tasks that the poll thread and the **sqlexec** thread perform in communicating with a client application.

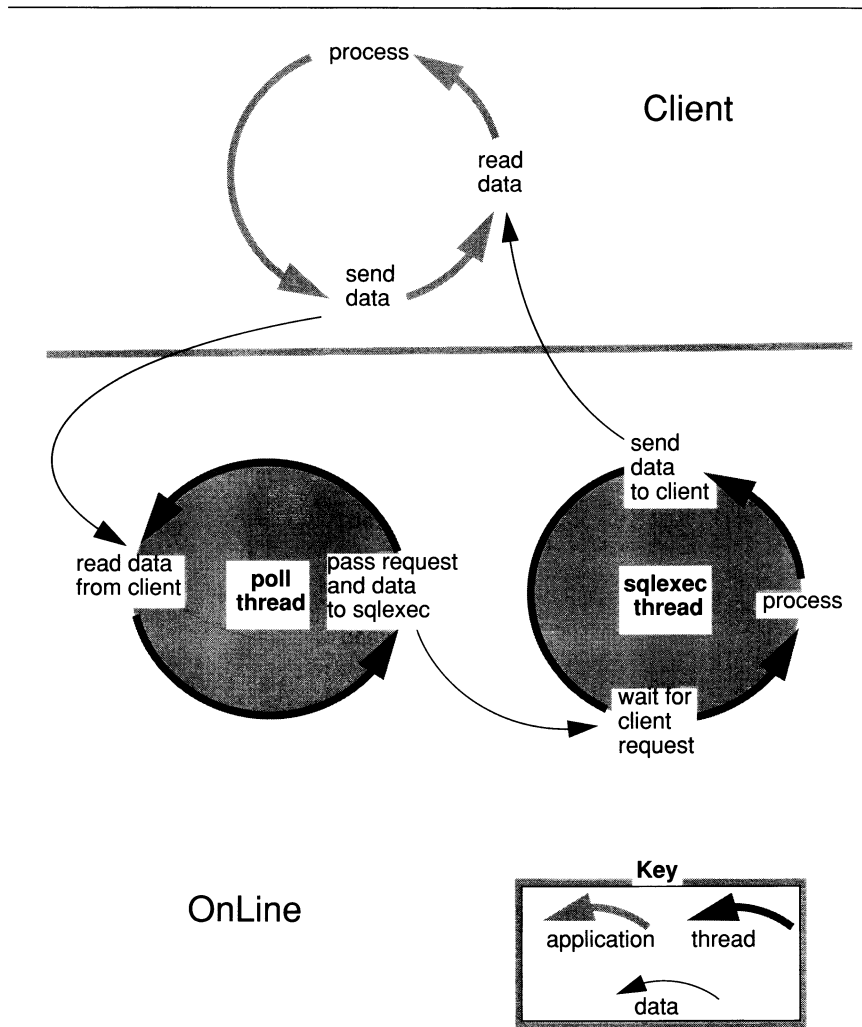


Figure 12-12 The roles of the poll and sqlxec threads in communicating with the client application

Starting Multiple Listen Threads

If **OnLine** is not able to satisfactorily service connection requests for a given protocol with a single port and corresponding listen thread, you can improve service for connection requests in the following two ways:

- Adding listen threads for additional ports
- Adding another network interface card

Adding Listen Threads for Additional Ports

As stated previously, **OnLine** 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 the following example defines two additional **dbservernames**, **soc_o12** and **soc_o13**, for the **OnLine** database server identified as **soc_o11**.

```
DBSERVERNAME      soc_o11
DBSERVERALIASES  soc_o12,soc_o13
```

Figure 12-13 Define multiple **dbservernames** for multiple connections of the same type

Once you define additional **dbservernames** for the database server, you must specify a protocol 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 12-15 cause **OnLine** to start three listen threads for the **onsoctcp** protocol, one for each of the ports defined.

dbservername	protocol	hostname	service name
soc_o11	onsoctcp	myhost	port1
soc_o12	onsoctcp	myhost	port2
soc_o13	onsoctcp	myhost	port3

Figure 12-14 **sqlhosts** file entries to listen to multiple ports for a single protocol

If you include a **NETTYPE** parameter for a protocol, it applies to all of the connections for that protocol. In other words, if a **NETTYPE** parameter exists for **onsoctcp** in the preceding example, it applies to all of the connections shown in Figure 12-15. In this example, **OnLine** runs one *poll* thread for the **onsoctcp**

protocol, unless the `NETTYPE` parameter specifies more. See “The `$INFORMIXDIR/etc/sqlhosts` File” on page 4-10 for more information about entries in the `sqlhosts` file.

Adding a Network-Interface Card

If the network-interface card for the host computer is unable to satisfactorily service connection requests, or if you want to connect **OnLine** 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 12-13, the `sqlhosts` file entries shown in Figure 12-15 cause **OnLine** to start three listen threads for the same protocol (as did the entries in Figure 12-15). 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**).

dbservername	protocol	hostname	service name
soc_ol1	onsoc tcp	myhost1	port1
soc_ol2	onsoc tcp	myhost1	port2
soc_ol3	onsoc tcp	myhost2	port1

Figure 12-15 Example of `sqlhosts` file entries to support two network-interface cards for the `onsoc tcp` protocol

Administration Virtual Processors

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.

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

OnLine 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. See the *INFORMIX-OnLine Dynamic Server Trusted Facility Manual* for more information about **OnLine** auditing.

Managing Virtual Processors

Chapter Overview	3
Setting Virtual Processor Configuration Parameters	3
Setting Virtual Processor Configuration Parameters Using ON-Monitor	3
Setting Virtual Processor Configuration Parameters Using a Text Editor	5
Starting and Stopping Virtual Processors	6
Adding Virtual Processors in On-Line Mode	7
Using onmode to Add Virtual Processors While OnLine Is in On-Line Mode	7
Using ON-Monitor to Add Virtual Processors While OnLine Is in On-Line Mode	7
Adding Network Virtual Processors	8
Dropping CPU Virtual Processors in On-Line Mode	9

Chapter Overview

This chapter describes how to set the configuration parameters that affect **INFORMIX-OnLine Dynamic Server** virtual processors. This chapter also tells you how to start and stop virtual processors.

See Chapter 12, “What Is the Dynamic Scalable Architecture?” for descriptions of the virtual-processor classes and for advice on determining how many virtual processors you should specify for each class.

Setting Virtual Processor Configuration Parameters

You can set the configuration parameters for **OnLine** virtual processors 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. See “Reinitializing Shared Memory” on page 15-14 for information on how to reinitialize shared memory.

Setting Virtual Processor Configuration Parameters Using ON-Monitor

To set the virtual processor configuration parameters using ON-Monitor, select Parameters from the main menu, and then select the perFormance option.

Figure 13-1 shows the full perFormance screen; the shaded entries set configuration parameters for OnLine virtual processors.

```

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]
CPU VPs                     [ 5]      Num of Read Ahead Pages    [ 50]
AIO VPs                     [ 1]      Read Ahead Threshold       [ 20]
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]

NETTYPE settings:
Protocol Threads Users VP-class
[ipcshm] [ 2] [ 5] [CPU]
[soctcp] [ 2] [ 5] [ ]

Are you running on a multiprocessor machine?
    
```

Figure 13-1 ON-Monitor perFormance screen

Figure 13-2 shows only the perFormance screen entries for configuring virtual processors. For each entry, it shows within a pair of brackets ([]), the name of the associated parameter in the ONCONFIG file.

```

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]
[soctcp] [NETTYPE]

```

Figure 13-2 Partial view of ON-Monitor perFormance screen showing the ONCONFIG parameter for each of the virtual processor entries

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.

See Figure 13-3 on page 13-6 for more information on the ONCONFIG parameters that are associated with `OnLine` virtual processors.

Setting Virtual Processor Configuration Parameters Using 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 13-3 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 35, “OnLine Configuration Parameters.”

Parameter	Purpose	Page
NUMCPUVPS	Specifies the number of CPU virtual processors	page 35-30
NUMAIOVPS	Specifies the number of AIO virtual processors	page 35-30
NETTYPE	Specifies parameters for network protocol threads (and virtual processors)	page 35-28
SINGLE_CPU_VP	Specifies that you are running a single CPU virtual processor	page 35-40
MULTIPROCESSOR	Specifies that you are running on a multiprocessor machine	page 35-27
AFF_NPROCS	Specifies the number of CPUs to which CPU virtual processors will be assigned (multiprocessor computers only)	page 35-8
AFF_SPROC	Specifies the first CPU (of AFF_NPROCS) to which a CPU virtual processor will be assigned	page 35-8
NOAGE	Specifies no priority aging of processes by the operating system	page 35-30

Figure 13-3 Table of ONCONFIG parameters for configuring virtual processors

Starting and Stopping Virtual Processors

When you start the **oninit** process to start **OnLine**, **oninit** starts the number and types of virtual processors that you have specified, directly and indirectly. You configure **OnLine** virtual processors primarily through ONCONFIG parameters and, for network virtual processors, through parameters in the **sqlhosts** file. See “Virtual Processor Classes” on page 12-16 for descriptions of the virtual-processor classes.

OnLine allows you to start a maximum of 1000 virtual processors.

Once **OnLine** is in on-line mode you can start additional virtual processors to improve performance, if necessary. See “Adding Virtual Processors in On-Line Mode” for information on how to do this.

While **OnLine** is in on-line mode, you can drop only virtual processors of the CPU class. See “Dropping CPU Virtual Processors in On-Line Mode” on page 13-9 for information on how to do this.

To terminate **OnLine** and thereby terminate all virtual processors, use the **-k** option of the **onmode** utility. See “Change OnLine Modes” on page 37-29 for more information on using the **-k** option of the **onmode** utility.

Adding Virtual Processors in On-Line Mode

While **OnLine** is in on-line mode, you can start additional virtual processors for the following classes: CPU, AIO, PIO, LIO, SHM, TLI, and SOC. You start additional virtual processors for these classes in one of the following two ways:

- Using the **-p** option of the **onmode** utility
- Using ON-Monitor

See “onmode: Mode and Shared-Memory Changes” on page 37-27 for the format of the **onmode** command.

Using onmode to Add Virtual Processors While OnLine Is in On-Line Mode

Use the **-p** option of the **onmode** command to add virtual processors while **OnLine** 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 is 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 lower case letters, as follows: **cpu**, **aio**, **pio**, **lio**, **shm**, **tli**, or **soc**. For example, if **OnLine** is currently running two virtual processors in the AIO class, either of the following commands starts four more.

```
% onmode -p 4 aio
```

or

```
% onmode -p +4 aio
```

The **onmode** utility starts the additional virtual processors immediately.

You can only add virtual processors to one class at a time. To add virtual processors for another class, you must run **onmode** again.

Using ON-Monitor to Add Virtual Processors While OnLine Is in On-Line Mode

To use ON-Monitor to add virtual processors while **OnLine** is in on-line mode, select Modes from the main menu and then select Add-Proc.

Figure 13-4 shows the ON-Monitor Add-Proc screen, which allows you to add virtual processors in the following classes: CPU, AIO, LIO, PIO and network.

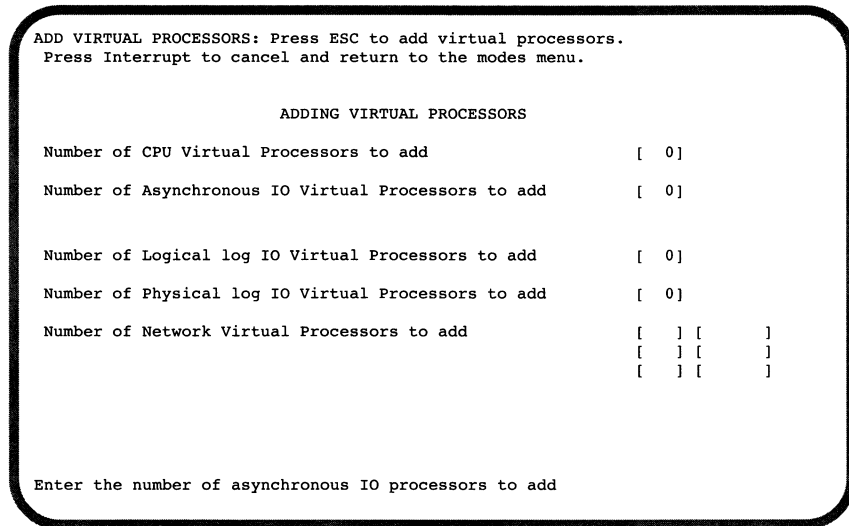


Figure 13-4 *The ON-Monitor Add-Proc screen allows you to add virtual processors while OnLine is in on-line mode.*

The Logical log and Physical log entries on the Add-Proc screen allow you to enter a number greater than 2 but **OnLine** will not start more than two virtual processors in either of these classes. **OnLine** 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 the protocol: `ipcshm`, `tlitcp`, `tlispx`, or `soctcp`.

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 **OnLine** is in on-line mode, and you have specified on the `NETTYPE` parameter that the poll threads run in the CPU class, **OnLine** does not start the new poll threads if no CPU virtual processors are available to run them.

Dropping CPU Virtual Processors in On-Line Mode

While **OnLine** is in on-line mode, you can use the **-p** option of the **onmode** utility to drop, or terminate, virtual processors of the CPU class. Following the **onmode** command, specify a negative number that is the number of CPU virtual processors you want to drop and then specify the CPU class in lower-case letters. For example, the following command drops two CPU virtual processors.

```
% onmode -p -2 cpu
```

You can only drop virtual processors of the CPU class while **OnLine** is in on-line mode.

If you attempt to drop a CPU virtual processor that is running a poll thread while **OnLine** is in on-line mode, you will receive the following message.

```
% onmode: failed when trying to change the number of cpu  
virtual processor by -<number>.
```

See “Should Poll Threads Run on CPU or Network Virtual Processors?” on page 12-25 for more information on CPU virtual processors and poll threads.

OnLine Shared Memory

Chapter Overview	5
What Is Shared Memory?	5
How OnLine Uses Shared Memory	6
How OnLine Allocates Shared Memory	8
How Much Shared Memory Does OnLine Use?	10
What Processes Attach to OnLine Shared Memory?	10
How a Client Attaches to the Communications Portion	10
Where the Client Attaches to the Communications Portion	11
How Utilities Attach to Shared Memory	11
How Virtual Processors Attach to Shared Memory	11
SERVERNUM Defines a Unique Key Value	12
SHMBASE Specifies Where to Attach the First Shared-Memory Segment	13
How Virtual Processors Attach Additional Shared- Memory Segments	13
Beware of the Shared-Memory Lower-Boundary Address	14
The Resident Portion of OnLine Shared Memory	15
Shared-Memory Header	17

Shared-Memory Internal Tables	18
Hash Tables	18
OnLine Buffer Table	18
OnLine Chunk Table	19
OnLine Dbspace Table	20
OnLine Lock Table	20
OnLine Page-Cleaner Table	21
OnLine Tblspace Table	21
OnLine Transaction Table	22
OnLine User Table	23
Shared-Memory Buffer Pool	23
Regular Buffers	23
Logical-Log Buffer	24
Physical-Log Buffer	25
Data-Replication Buffer	25
The Virtual Portion of OnLine Shared Memory	25
How OnLine Manages the Virtual Portion of Shared Memory	26
How to Specify the Size of the Virtual Portion of Shared Memory	26
What Is in the Virtual Portion of Shared Memory	26
Big Buffers	27
Session Data	27
Thread Data	27
Dictionary Cache	28
Sorting Memory	28
Stored Procedures Cache	29
Global Pool	29
The Communications Portion of OnLine Shared Memory	29
Concurrency Control	30
Shared-Memory Mutexes	30
Shared-Memory Buffer Locks	31
Types of Buffer Locks	31
How OnLine Threads Access Shared Buffers	32
OnLine LRU Queues	32
LRU Queue Components	32
LRU Queues and Buffer-Pool Management	33
Limiting the Number of Pages Added to the MLRU Queues	34
When MLRU Cleaning Ends	34
Read Ahead	35

How an OnLine Thread Accesses a Buffer Page	36
Identify the Page	36
Determine the Level of Lock Access	36
Try to Locate the Page in Shared Memory	36
Or, Locate a Buffer and Read Page from Disk	37
Lock the Buffer If Necessary	37
Release the Buffer Lock and Wake a Waiting Thread	37
How OnLine Flushes Data to Disk	39
Three Events Prompt Flushing of the Regular Buffers	39
Rule: Before-Images Are Flushed First	39
Flushing the Physical-Log Buffer	39
Three Events Prompt Flushing of the Physical-Log Buffer	40
When the Physical-Log Buffer Becomes Full	40
How OnLine Synchronizes Buffer Flushing	41
How OnLine Makes Sure the Physical-Log Buffers Are Flushed First	41
Types of Writes that Prompt Flushing Activity	42
Foreground Write	43
LRU Write	43
Chunk Write	43
Flushing the Logical-Log Buffer	44
Five Events Prompt Flushing of the Logical-Log Buffers	44
When the Logical-Log Buffer Becomes Full	44
After a Transaction Is Prepared or Terminated in a Database with Unbuffered Logging	45
When a Session That Uses Nonlogging Databases or Unbuffered Logging Terminates	46
When a Checkpoint Occurs	46
When a Page Is Modified That Does Not Require a Before-Image in the Physical-Log File	46
How OnLine Achieves Data Consistency	46
Critical Sections	46
OnLine Checkpoints	47
Five Events Initiate a Checkpoint	47
Main Events During a Checkpoint	48
Checkpoint Is Critical to Fast Recovery	49
OnLine Timestamps	50
Timestamps on Disk Pages	50
Timestamps on Blob Pages	50
Blob Timestamps with Dirty Read and Committed Read Isolation Levels	51

Writing Data to a BlobSpace	52
BlobPages Do Not Pass Through Shared Memory	52
Blobs Are Created Before the Data Row Is Inserted	53
BlobPage Buffers Are Created for the Duration of the Write	53

Chapter Overview

This chapter describes the content of **INFORMIX-OnLine Dynamic Server** shared memory, the factors that determine the sizes of shared-memory areas, and how data moves into and out of shared memory. See Chapter 15, “Managing OnLine Shared Memory,” for information on how to change the **OnLine** configuration parameters that determine shared-memory allocations.

What Is Shared Memory?

Shared memory is an operating-system feature that allows **OnLine** threads and processes to share data by sharing access to pools of memory. **OnLine** 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 **OnLine** to reduce overall memory usage because the participating processes—in this case, virtual processors—do not need to maintain private copies of the data that is in shared memory.

Shared memory reduces disk I/O because buffers, which are managed as a common pool, are flushed on a database-server-wide basis instead of a per-process basis. Furthermore, a virtual processor is often able to 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 OnLine Uses Shared Memory

OnLine uses shared memory for the following purposes:

- To enable OnLine virtual processors and utilities to share data
- To provide a fast communications channel for local client applications

Figure 14-1 on page 14-7 illustrates how OnLine uses shared memory.

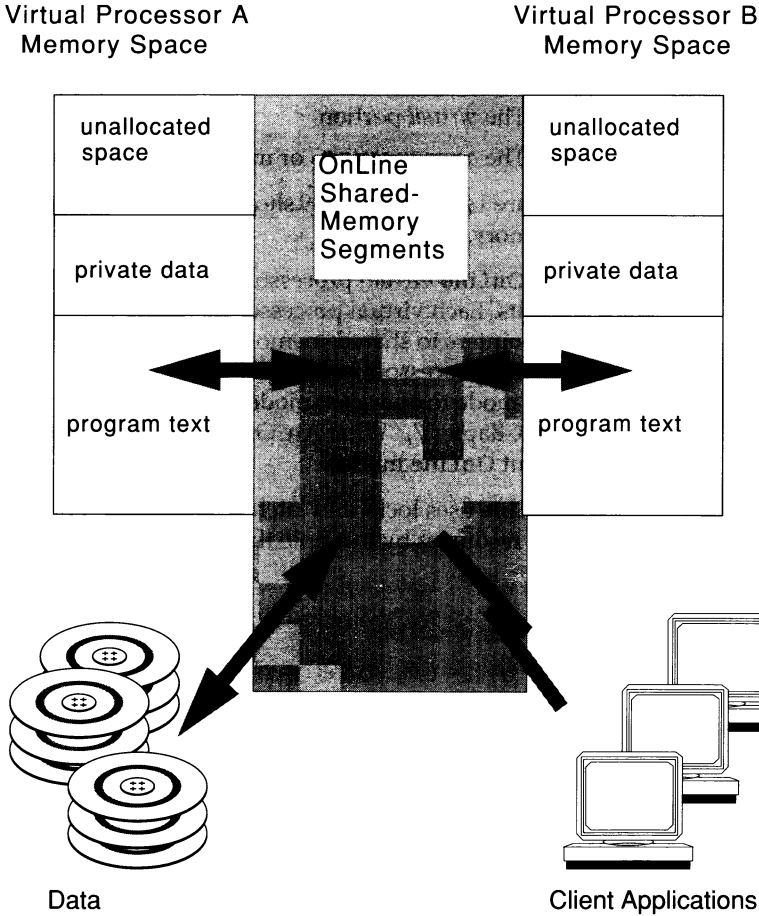


Figure 14-1 How OnLine uses shared memory

How OnLine Allocates Shared Memory

When **OnLine** initializes shared memory, it acquires shared-memory segments for the following three portions:

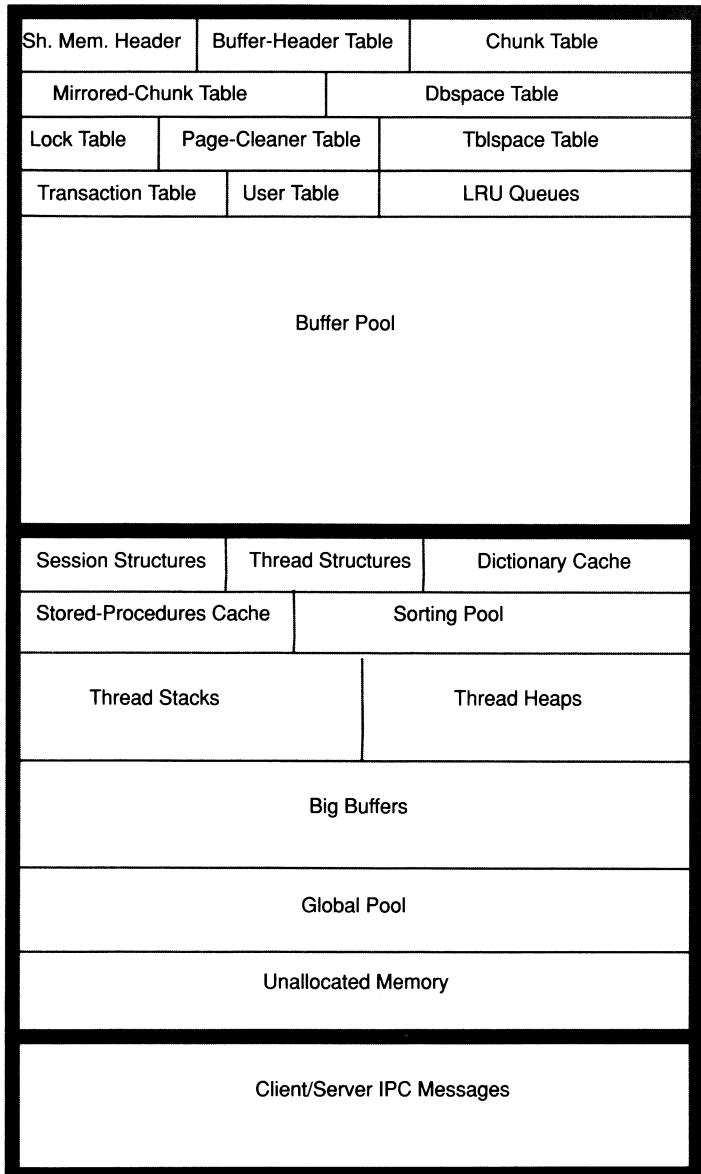
- The *resident* portion
- The *virtual* portion
- The *communications* or message system portion

Figure 14-2 on page 14-9 shows the contents of each portion of **OnLine** shared memory.

All **OnLine** 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 **OnLine** from off-line mode to quiescent mode, or from off-line mode directly to on-line mode. See Chapter 7, “What Are OnLine Operating Modes?” for more information about **OnLine** modes.

OnLine uses locks and latches to manage concurrent access to shared memory resources by multiple threads.

Resident Portion



Virtual Portion

Communications Portion

Figure 14-2 Contents of OnLine shared memory

How Much Shared Memory Does OnLine Use?

Each portion of **OnLine** shared memory consists of one or more operating-system segments of memory, each one divided into a series of blocks, eight kilobytes (8k) in size and managed by a bit map.

The header-line output by the **onstat** utility contains the size of **OnLine** shared memory, expressed in kilobytes. See “onstat: Monitor OnLine Operation” on page 37-46 for information on how to use the **onstat** utility.

You can set the SHMTOTAL parameter in the ONCONFIG file to limit the amount of memory overhead that **OnLine** can place on your system. The SHMTOTAL parameter specifies the total amount of virtual memory that **OnLine** can use for all memory allocations. Applications might fail, however, if **OnLine** needs more memory than the amount set in SHMTOTAL. If this condition occurs, **OnLine** displays the following message in the message log:

```
size of resident + virtual segments x + y > z
total allowed by configuration parameter SHMTOTAL
```

What Processes Attach to OnLine Shared Memory?

The following processes attach to **OnLine** shared memory:

- Client application processes that communicate with **OnLine** through the shared-memory communications portion (ipcshm protocol)
- **OnLine** virtual processors
- **OnLine** utilities

The following sections describe how each type of process attaches to **OnLine** shared memory.

How a Client Attaches to the Communications Portion

Client application processes that communicate with **OnLine** through shared memory (ipcshm protocol) attach transparently to the communications portion of **OnLine** shared memory. System library functions that are automatically compiled into the application enable it to attach to the communications portion of **OnLine** shared memory. See Chapter 4, “Configuring Connectivity,” and “Network Virtual Processors” on page 12-24 for information on specifying a shared-memory connection.

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 implementation-specific. If the client application attaches to other shared-memory segments (not **OnLine** shared memory), the user can set the `INFORMIXSHMBASE` environment variable to specify the address at which to attach the **OnLine** shared-memory communications segments. By specifying the address at which to address the shared-memory communications segments, you can prevent **OnLine** from colliding with the other shared-memory segments that your application uses. See Chapter 4 of *Informix Guide to SQL: Reference* for information on how to set the `INFORMIXSHMBASE` environment variable.

How Utilities Attach to Shared Memory

OnLine utilities such as `onstat`, `onmode`, and `ontape` attach to **OnLine** 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 **OnLine**. The file is removed when **OnLine** terminates.

How Virtual Processors Attach to Shared Memory

OnLine virtual processors attach to shared memory during initialization. During this process, **OnLine** 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 **OnLine**, if any, on the same computer.

OnLine 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. Attach additional shared-memory segments, if required, to be contiguous with the first segment.

The following sections describe how **OnLine** uses the values of the `SERVERNUM` and `SHMBASE` configuration parameters in the process of attaching shared-memory segments.

SERVERNUM Defines a Unique Key Value

OnLine uses the `ONCONFIG` parameter `SERVERNUM` to calculate a unique key value for its shared-memory segments. All virtual processors within a single **OnLine** 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 $(\text{SERVERNUM} * 65536)$ is the same as multiplying `SERVERNUM` by hexadecimal 10000.

When more than one **OnLine** 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 65536.

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.

SHMBASE Specifies Where to Attach the First Shared-Memory Segment

The `SHMBASE` parameter in the `ONCONFIG` file specifies the virtual address where each **OnLine** 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 enables all virtual processors within

the same **OnLine** instance to reference the same locations in shared memory without needing to calculate shared-memory addresses. All shared-memory addresses for an instance of **OnLine** 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 will keep 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, it is possible the data segment could grow into the shared-memory segment.
- Some versions of UNIX require the user to specify a SHMBASE 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 **OnLine** 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. **OnLine** 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 **OnLine** 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 $(\text{SERVERNUM} * 65536) + \text{shmkey}$, **OnLine** can request up to 65,536 shared-memory segments before it could request a shared-memory key value used by another **OnLine** instance on the same computer.

Beware of 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 14-3 on page 14-15 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 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.

OnLine receives errors when this situation occurs. To correct the problem, check the UNIX kernel parameter that specifies the low-boundary address or reconfigure the kernel to allow larger shared-memory segments. See "4 The Role of the Shared-Memory Lower-Boundary Address" on page 15-5 for a description of the UNIX kernel parameter.

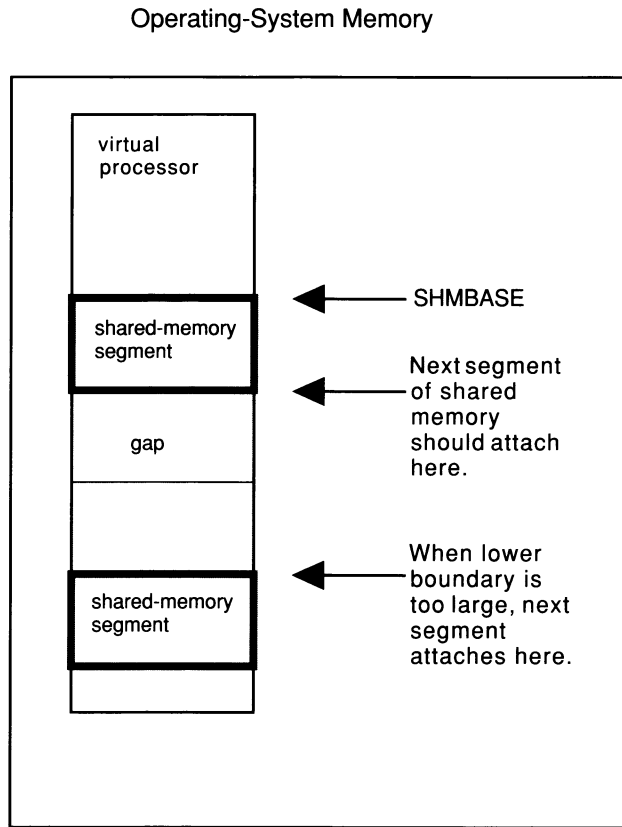


Figure 14-3 *If the shared-memory lower-boundary address is set incorrectly, a gap between shared-memory segments can result*

The Resident Portion of OnLine 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 **OnLine** shared memory stores the following data structures that do not change in size:

- Shared-memory header
- Internal tables
- Buffer pool

OnLine requests that the operating system keep this resident portion resident 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 one

Figure 14-4 on page 14-17 illustrates the contents of the resident portion of shared memory.

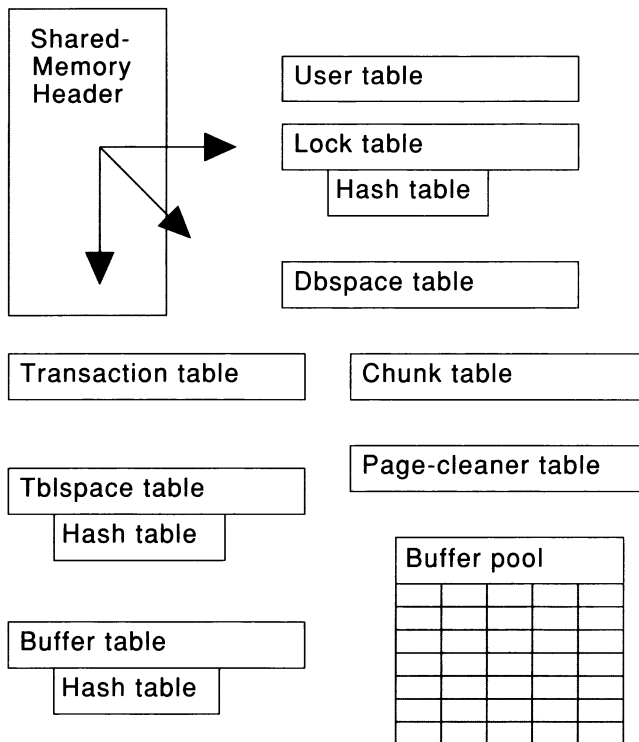


Figure 14-4 The resident portion of shared memory

Shared-Memory Header

The shared-memory header contains a description of all other structures in **OnLine** shared memory, including internal tables and the **OnLine** 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. The administrator cannot tune the size of the header.

Shared-Memory Internal Tables

OnLine shared memory contains nine internal tables that track shared-memory resources. Three of these nine tables are paired with hash tables. The shared-memory internal tables are as follows:

- Buffer table and associated hash table
- Chunk table
- Dbspace table
- Lock table and associated hash table
- Page-cleaner table
- Tblspace table and associated hash table
- Transaction table
- User table

Hash Tables

Hashing is a technique that permits rapid lookup in tables where items are added unpredictably. The following three **OnLine** 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.

OnLine Buffer Table

The buffer table tracks the address and status of the individual buffers in the shared-memory pool. When a buffer is used, it contains an image of a data or index page from disk. See “What Is a Page?” on page 10-9 and “Structure and Storage of a Dbspace Page” on page 40-30 for more information on the purpose and content of a disk page.

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. **OnLine** 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.

The minimum number of **OnLine** buffers is based on the number of **OnLine** user threads, specified as `USERTHREADS` in the `ONCONFIG` file. You must 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 512 kilobytes. See “Monitoring Buffers” on page 29-15 for information on how to monitor **OnLine** buffers. See “BUFFERS” on page 35-8 for information on how to specify the number of buffers available to **OnLine**.

Each **OnLine** buffer has one entry in the buffer table.

Buffer-Table Hash Table

OnLine 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`.

OnLine Chunk Table

The chunk table tracks all chunks in the **OnLine** database server. If mirroring has been enabled, a corresponding mirror chunk table is also created when shared memory is initialized. The mirror chunk table tracks all mirror chunks.

The chunk table in shared memory contains information that enables **OnLine** 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 `blob`space. See “Monitor Chunks” on page 29-46 for information on monitoring chunks.

The maximum number of entries in the chunk table is equal to the value of `CHUNKS`, as specified in the `ONCONFIG` file. The maximum number of chunks may be limited by the maximum number of file descriptors that the

operating system allows per process. (This is often specified by a kernel configuration parameter.) See “CHUNKS” on page 35-9 for information on how to specify the number of chunks available to **OnLine**.

OnLine Dbspace Table

The dbspace table tracks both dbspaces and blobspaces in the **OnLine** database server. The dbspace-table information includes the following information about each dbspace in the **OnLine** configuration:

- 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. See “Monitoring Disk Usage” on page 29-46 for information on monitoring dbspaces.

The maximum number of entries in the dbspace table is equal to the value of the DBSPACES parameter in the ONCONFIG file, which specifies the maximum number of dbspaces permitted in **OnLine**. See Chapter 15, “Managing OnLine Shared Memory,” for information on setting the DBSPACES parameter. See “DBSPACES” on page 35-13 for information on specifying the number of dbspaces available to **OnLine**.

OnLine 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. A single session is limited to 32 concurrent, explicit table locks. Refer to the *Informix Guide to SQL: Tutorial* for an explanation of locking and the SQL statements associated with locking.

The information stored in the table describes the lock. The lock description includes the following four items:

- 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

The maximum number of entries in the lock table is specified by the **LOCKS** parameter in the **ONCONFIG** file. See “LOCKS” on page 35-19 for information on specifying the number of locks available to **OnLine** sessions.

See “Monitoring Locks” on page 29-23 for information on monitoring locks.

When Is a Byte Lock Generated?

A byte lock is only generated if you are using **VARCHAR** data types. 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).

OnLine 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. See “CLEANERS” on page 35-11 for advice on how many page-cleaner threads to specify.

The page-cleaner table always contains 32 entries, regardless of the number of page cleaner threads specified by the **CLEANERS** parameter in the **ONCONFIG** file.

See the **-F** option under “onstat: Monitor OnLine Operation” on page 37-46 for information on monitoring the activity of page-cleaner threads.

OnLine Tblspace Table

The tblspace table tracks all active tblspaces in the **OnLine** system. An active tblspace is one that is currently in use by an **OnLine** session. Each active table accounts for one entry in the tblspace table. Active tblspaces include database tables, temporary tables, and internal control tables, such as system catalog tables. Each tblspace table entry includes header information about the tblspace, the tblspace name, and pointers to the tblspace tblspace in the root

dbspace on disk. (Do not confuse the shared-memory active tblspace table with the tblspace tblspace.) See “Monitoring Tblspaces and Extents” on page 29-50 for information on monitoring tblspaces.

The maximum number of entries in the tblspace table is specified by the TBLSPACES parameter in the ONCONFIG file. If a session attempts to open an additional table after all entries are used, an error is returned. See “TBLSPACES” on page 35-44 for information on specifying the maximum number of active tblspaces available to **OnLine**.

Entries in the tblspace table are tracked in an associated hash table.

Hash Table Entries

The number of entries in the tblspace hash table is based on the number of allocated tblspaces (specified as TBLSPACES in the ONCONFIG file). The maximum number of hash values is the largest power of two that is less than the value specified by the expression (TBLSPACES divided by 4). For example, if you set TBLSPACES to 500 then the value of (TBLSPACES divided by 4) is 125. Therefore, the maximum number of hash values would be 5 because 2^5 is the highest power of 2 that has a value (64) less than 125.

OnLine Transaction Table

The transaction table tracks all transactions in the **OnLine** 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, see the product documentation for **INFORMIX-TP/XA**.

Some information that had been tracked in the user table in earlier releases is now tracked in the transaction table. Tracking information derived from the transaction table appears in the **onstat -x** display. See “Monitoring Transactions” on page 29-33 for an example of the output displayed by **onstat -x**.

The number of entries in the transaction table is based on the TRANSACTIONS parameter in the ONCONFIG file.

See the *Informix Guide to SQL: Tutorial* and the *Informix Guide to SQL: Reference* for more information on transactions and the SQL statements that you use with transactions.

OnLine User Table

The user table tracks all user threads. These threads include a thread to accept requests from ON-Monitor, a thread to accept requests from the **onmode** utility, the threads that are used in recovery, and page-cleaner threads.

The maximum number of entries in the user table is equal to the number of user threads permitted on this **OnLine** system, specified by the **USER-THREADS** parameter in the **ONCONFIG** file. If **OnLine** needs more user threads than you allocated on the **USERTHREADS** parameter, it returns an error.

You can monitor user threads using the **onstat -u** command.

Shared-Memory Buffer Pool

The **OnLine** 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, that function solely to ensure the physical and logical consistency of **OnLine** data:

- Logical-log buffer
- Physical-log buffer

If you have implemented data replication, the buffer pool in resident shared memory also contains a data-replication 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.

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**:

- **-b** and **-B** options display general buffer information.
- **-R** displays LRU queue statistics.
- **-X** displays information about **OnLine** I/O threads that are waiting for buffers.

See “OnLine LRU Queues” on page 14-32 for a description of how LRU queues work. See “Mutexes” on page 12-16 for a description of mutexes.

You specify the number of regular buffers in the buffer pool on the **BUFFERS** parameter in the **ONCONFIG** file. See “BUFFERS” on page 35-8 for information on specifying the number of **BUFFERS** available to **OnLine**.

How Big Is a Regular Buffer?

Each regular buffer is the size of one **OnLine** page. In general, **OnLine** performs I/O in full-page units, the size of a regular buffer. The two exceptions are I/O performed from big buffers and I/O performed from blob space buffers. To determine the **OnLine** 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 **OnLine** page size is the last entry on the page.

Logical-Log Buffer

OnLine uses the logical log to store a record of changes to **OnLine** data since the last archive. The logical log stores records that represent logical units of work for **OnLine**. It contains the following five types of log records:

- 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

There are three logical-log buffers. Triple-buffering permits the logical-log (LIO) virtual processor to write to two active buffers while one buffer is flushed to disk. **OnLine** uses three buffers for the logical log because it is flushed more often than the physical log, especially when unbuffered logging is used. Two active buffers provide adequate time to flush the third buffer to disk. See “Flushing the Logical-Log Buffer” on page 14-44 for a description of how **OnLine** flushes the logical-log buffer.

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, blobs in dbspaces). See “**LOG-BUFF**” on page 35-20 for the possible values that you can assign to this parameter.

Physical-Log Buffer

OnLine 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 **OnLine** to restore consistency to its databases after a system failure.

The physical-log buffer is actually two buffers. Double buffering permits **OnLine** processes to write to the active physical-log buffer while the other buffer is being flushed to the physical log on disk. See “Flushing the Physical-Log Buffer” on page 14-39 for a description of how **OnLine** flushes the physical-log buffer.

See “Monitoring the Physical-Log File” on page 29-40 for information on monitoring the physical-log file.

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, **OnLine** rounds the size down to the nearest value that is evenly divisible by the page size. When the buffer fills, **OnLine** flushes the buffer to the physical-log file on disk. So the size of the buffer determines how frequently **OnLine** needs to flush it to disk. See “`PHYSBUFF`” on page 35-32 for more information on this parameter.

Data-Replication Buffer

Data replication requires two instances of **OnLine**, a primary **OnLine** and a secondary **OnLine**, running on two computers. If you implement data replication for your **OnLine** database server, **OnLine** holds logical-log records in the data-replication buffer before sending them to the secondary **OnLine**. The data-replication buffer is always the same size as the logical-log buffer. See the preceding section, “Logical-Log Buffer,” for information on the size of the logical-log buffer. See “How Does Data Replication Work?” on page 25-8 for more information on how the data-replication buffer is used.

The Virtual Portion of OnLine Shared Memory

The virtual portion of shared memory is expandable by **OnLine** and can be paged out to disk by the operating system. As **OnLine** executes, it automatically attaches additional operating system segments, as needed, to the virtual portion.

How OnLine Manages the Virtual Portion of Shared Memory

OnLine uses memory *pools* to track memory allocations that are of a similar type and size. Keeping related memory allocations in a pool helps to reduce memory fragmentation. It also enables **OnLine** to free a large allocation of memory at one time, as opposed to freeing each piece that makes up the pool. If there is insufficient memory available in a pool to satisfy a request, **OnLine** adds memory from unused memory in the virtual segment. If **OnLine** cannot find enough memory in the virtual segment, it dynamically adds another segment to the virtual portion.

OnLine allocates virtual shared memory for each of its subsystems (session pools, stacks, heaps, control blocks, the system catalog and stored procedure caches, sort pools, and message buffers) from pools that track free space through a linked list. When **OnLine** allocates a portion of memory, it first searches the pool free-list for a *fragment* of sufficient size. If none is found, new blocks are brought into the pool from the virtual portion. When memory is freed, it goes back to the pool as a free fragment and remains there until the pool is destroyed. When **OnLine** starts a session for a client application, for example, it allocates memory from the session pool, the stack pool, and the heap pool. When the session terminates, the allocated memory is returned to these pools as free fragments.

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.

See “SHMVIRTSIZE” on page 35-39, “SHMADD” on page 35-37, and “Add-
ing a Segment to the Virtual Portion of Shared Memory” on page 15-16 for
more information on determining the size of virtual shared memory.

What Is in 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 the size of 32 pages. **OnLine** allocates big buffers to improve performance on large reads and writes.

OnLine uses a big buffer whenever it writes to disk multiple pages that are physically contiguous. For example, **OnLine** tries to use a big buffer to perform a series of sequential reads or to write a dbspace blob into shared memory. After disk pages are read into the big buffer, they are immediately allocated to regular buffers in the buffer pools. **OnLine** also uses big buffers in sorted writes and in chunk writes during checkpoints. See “onstat: Monitor OnLine Operation” on page 37-46 for information on monitoring the **OnLine** use of big buffers.

Session Data

When a client application requests a connection to **OnLine**, **OnLine** 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.

OnLine allocates memory for session structures as needed.

Thread Data

When a client connects to **OnLine**, in addition to starting a session, **OnLine** starts a primary session thread and creates a *thread control block* (tcb) for it in shared memory.

OnLine also starts internal threads on its own behalf and creates thread control blocks for these. When **OnLine** 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. See “Context Switching” on page 12-10 for more information on the thread control block and how it is used.

OnLine allocates memory for thread control blocks as needed.

Stacks

Each thread in **OnLine** has its own stack area in the virtual portion of shared memory. See “Stacks” on page 12-12 for a description of how **OnLine** threads use stacks. See “Monitoring Sessions and Threads” on page 29-29 for information on how to monitor the size of the stack for a session.

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`. See “`STACKSIZE`” on page 35-40 for information and a warning on setting the size of the stack.

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. See the description of the `INFORMIXSTACKSIZE` environment variable in the *Informix Guide to SQL: Reference* for information on how to override the stack size for a particular user.

It is safer to alter the size of stack space by using the `INFORMIXSTACKSIZE` environment variable because it 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 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, **OnLine** 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 comprise the dictionary cache.

The size of the dictionary cache is not configurable.

Sorting Memory

The amount of virtual shared memory that **OnLine** allocates for a sort depends on the number of rows to be sorted and the size of the row. The maximum amount of shared memory that **OnLine** allocates for a sort is 5 mega-

bytes. If the PSORT_NPROCS environment variable is set, requesting a parallel sort, the amount of memory allocated is divided by the number of threads that will do the sort. See “Psort (Parallel-Sort) Package” on page 30-8 for more information on parallel sorts and the PSORT_NPROCS environment variable.

To account for the amount of virtual shared memory that **OnLine** might need for sorting, estimate the maximum number of sorts that might occur concurrently and multiply it 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 **OnLine** needs for sorting as follows:

$$30 \text{ sorts} * 200 \text{ bytes} * 400 \text{ rows} = 2,400,000 \text{ (or approx. 2.4 meg)}$$

Stored Procedures Cache

When a session needs to access a stored procedure for the first time, **OnLine** reads the stored procedure from the system catalog tables. **OnLine** 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 **OnLine**. For example, the global pool contains the message queues where poll threads for network protocols (soctcp, tlitcp, ipx/spx) deposit messages from clients. The **sqlxec** threads pick up the messages from here and process them.

The Communications Portion of OnLine Shared Memory

The communications, or message system, portion of shared memory is allocated when shared memory is initialized. The communications portion contains the message buffers for local client applications that use shared memory to communicate with **OnLine**.

The size of the communications portion of shared memory is based on the number of threads specified for the `ipcshm` protocol. The size of the communications portion is approximately 12 kilobytes multiplied by the number of threads specified. The number of threads is the lesser of the following two values:

- The value of the `USERTHREADS` configuration parameter
- The number specified in the `users` entry of the `NETTYPE` parameter for the `ipcshm` protocol, if a parameter is present and the number is lower than the value of `USERTHREADS`.

See “How a Client Attaches to the Communications Portion” on page 14-10 for information about how a client attaches to the communications portion of shared memory.

Concurrency Control

OnLine threads that run on the same virtual processor, and on separate virtual processors, share access to resources in shared memory. When an **OnLine** thread writes to shared memory, it uses mechanisms called *latches* and *locks* to prevent other threads from simultaneously writing to the same area. A latch (called a *mutex* in **OnLine**) 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

OnLine uses latches, 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 an **OnLine** thread can modify a shared-memory resource (such as a table), 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 it held by another thread, the incoming thread must wait for the mutex to be released.

For example, two **OnLine** threads can attempt to access the same slot in the chunk table, but only one can acquire the mutex associated with the 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.

See “Monitoring Latches” on page 29-22 for information on monitoring mutexes (which are also referred to as latches in the output from the monitoring tools).

Shared-Memory Buffer Locks

A primary benefit of shared memory is the ability of **OnLine** threads to share access to disk pages stored in the shared-memory buffer pool. **OnLine** maintains thread isolation while achieving this increased concurrency through a strategy for locking the data buffers.

Types of Buffer Locks

OnLine 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 **OnLine** thread isolation during execution.

See “Monitoring Locks” on page 29-23 for information on how to monitor the use of locks.

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 page 14-31

The Share Lock

A buffer is in share mode, or has a share lock, if multiple **OnLine** 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 threads requesting access to the buffer are placed on 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 “OnLine Lock Table” on page 14-20 and “Shared-Memory Buffer Locks” on page 14-31.

How OnLine Threads Access Shared Buffers

OnLine threads access shared buffers through a system of queues, using latches and locks to synchronize access and protect data.

OnLine LRU Queues

Each regular buffer is tracked through several linked lists of pointers to the buffer table. 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 **OnLine** 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:

- 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 14-5 on page 14-33 illustrates the structure of the LRU queues.

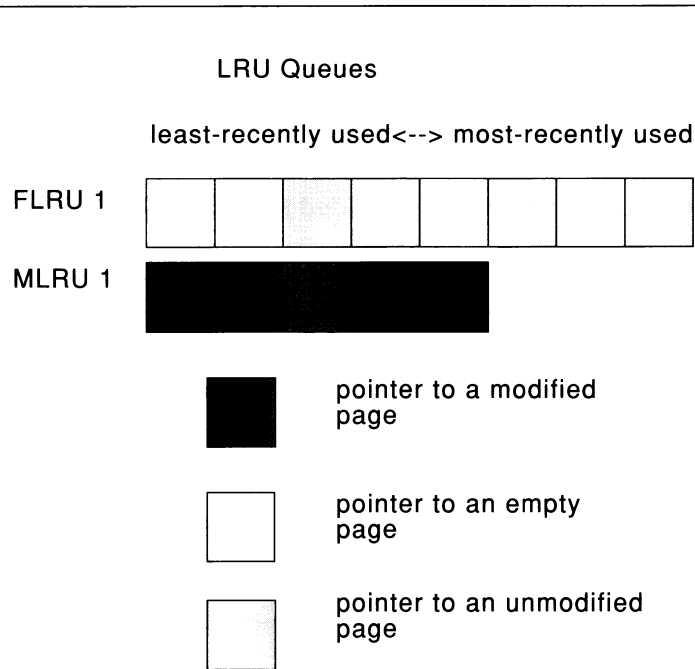


Figure 14-5 LRU queues

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, **OnLine** 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, **OnLine** 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. See “Monitoring Buffer-Pool Activity” on page 29-19 for information on how to monitor LRU queues.

Limiting the Number of Pages Added to the MLRU Queues

Periodically, the modified buffers in an MLRU queue are written (flushed) to disk by the page-cleaner threads. You can specify the point at which cleaning begins using 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 default value of LRU_MAX_DIRTY is 60, meaning that 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. See “How OnLine Flushes Data to Disk” on page 14-39 for more information on how OnLine performs buffer-pool flushing.

Figure 14-6 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 14-6 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 percentage of buffers in an MLRU queue. The default value of LRU_MIN_DIRTY is 50, mean-

ing that page cleaning is no longer required when 50 percent of the buffers in an LRU queue are modified (that is, in the MLRU queue). In practice, page cleaning can continue beyond this point as directed by the page-cleaner threads.

Figure 14-7 shows how the value of `LRU_MIN_DIRTY` is applied to the LRU queue to specify the acceptable percentage of buffers in an MLRU queue and the point at which page cleaning ends:

```

BUFFERS specified as 8000
LRUS specified as 8
LRU_MIN_DIRTY specified as 50

The acceptable number of buffers in the MLRU queue and the point
at which page cleaning can end is equal to LRU_MIN_DIRTY

Buffers per LRU queue = (8000/8) = 1000

Acceptable number of buffers in MLRU queue and the point at which
page cleaning can end: 1000 x .050 = 500

```

Figure 14-7 How LRU_MIN_DIRTY specifies the point at which page cleaning can end

See “How OnLine Flushes Data to Disk” on page 14-39 for more information on how **OnLine** flushes the buffer pool.

Read Ahead

For sequential table or index scans, you can configure **OnLine** 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.

OnLine 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 **OnLine** does a read-ahead.

The `RA_THRESHOLD` parameter specifies the number of unprocessed pages in memory that cause **OnLine** to do another read-ahead. For example, if `RA_PAGES` is 10 and `RA_THRESHOLD` is 4, **OnLine** reads ahead 10 pages when 4 pages remain to be processed in the buffer. See “Monitoring Shared-Memory Profile” on page 29-13 for an example of the output that the `onstat`

-p command produces to enable you to monitor **OnLine** use of read-ahead. See “-p Option” on page 37-61 under the heading “onstat: Monitor OnLine Operation.”

How an OnLine Thread Accesses a Buffer Page

OnLine uses shared-lock buffering to allow more than one **OnLine** thread to concurrently access the same buffer in shared memory. **OnLine** 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.

The process of accessing a data buffer consists of the following seven 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 with the buffer, release the lock.
7. Wake waiting threads with compatible lock access types, if any exist.

Identify the Page

OnLine threads request a specific data row and **OnLine** searches for the page that contains the row.

Determine the Level of Lock Access

Next **OnLine** 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 associated with the buffer table. Then, it searches the hash table to see if an entry matches the requested page. If it 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 **OnLine** to use read-ahead, **OnLine** performs a read-ahead request when the number of pages specified by the `RA_THRESHOLD` parameter remain to be processed in memory.

Or, 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. **OnLine** 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 **OnLine** to use read-ahead, **OnLine** 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 and, if any threads are waiting for the buffer, wakes one up. This procedure varies, however, depending on whether the releasing thread modified the buffer.

When the Buffer Is *Not* Modified

If the **OnLine** 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 looks to see if other **OnLine** 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 timestamp in the buffer header so that the timestamp on the buffer page and the timestamp 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 OnLine 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 **OnLine** 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. **OnLine** always runs at least one page-cleaner thread. If **OnLine** is configured for more than one page-cleaner thread, the LRU queues are divided among the page cleaners for more efficient flushing. See “CLEANERS” on page 35-11 for information on specifying how many page-cleaner threads **OnLine** runs.

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.

Three Events 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 and no unmodified buffers are available.
- **OnLine** needs to execute a checkpoint.

Rule: Before-Images Are Flushed First

The overriding rule of buffer flushing is this: 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 containing modified pages. Therefore, even when a shared-memory buffer page needs to be flushed because a user thread is trying to acquire a buffer and none are 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

OnLine 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 writ-

ten 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 is required for the fast-recovery feature.

Both the physical-log buffer and the physical log contribute toward maintaining the physical and logical consistency of **OnLine** data. See Chapter 20, “What Is Physical Logging?,” for a description of physical logging and Chapter 22, “What Is Fast Recovery?,” for a description of fast recovery.

Three Events 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.

OnLine uses only one of the two physical-log buffers at a time. This buffer is the current physical-log buffer. Before **OnLine** 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.

Once 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 Seventy-Five 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. See “Critical Sections” on page 14-46 for a description of a critical section.

If the Log Is Less Than Seventy-Five 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 releasing the mutex, the thread 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. If the before-image spans multiple pages, multiple pages are written to the physical-log buffer, one page at a time.

How OnLine Synchronizes Buffer Flushing

When OnLine 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.

How OnLine Makes Sure the 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.

How is this done? The answer is timestamp comparison.

OnLine stores a timestamp 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 timestamp in the modified buffer with the timestamp that indicates the point when the physical-log buffer was last flushed.

If the timestamp on the page in the buffer pool is equal to or more recent than the timestamp 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 timestamp 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 timestamp 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. (See “Sleep Queues” on page 12-14.) You can tune the page-cleaning parameters (`LRU_MIN_DIRTY` and `LRU_MAX_DIRTY`) to influence the frequency of buffer flushing. See “LRU Queues and Buffer-Pool Management” on page 14-33 for a description of how these parameters determine when page cleaning begins and ends.

Types of Writes that Prompt Flushing Activity

OnLine provides you with information about the specific condition that prompted buffer-flushing activity by defining three types of **OnLine** writes and counting how often each write occurs:

- Foreground write
- LRU write
- Chunk write

If you implement mirroring for **OnLine**, data is always written to the primary chunk first; then, the write is repeated on the mirror chunk. Writes to a mirror chunk are included in the counts. See “Monitoring Buffer-Pool Activity” on page 29-19 for more information on monitoring the types of writes that **OnLine** performs.

Foreground Write

If a user thread searches through the FLRU queues and cannot locate an empty or unmodified buffer, pages must be flushed to make space. (See “LRU Queues and Buffer-Pool Management” on page 14-33.) If the thread must perform buffer flushing just to acquire a shared-memory buffer, performance can suffer.

Page flushes that the user thread requests are called *foreground writes*. Foreground writes should be avoided. If you find that foreground writes are occurring, 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

A foreground write signals that page cleaning is needed. Once alerted, the CPU virtual processor wakes the page cleaner threads to begin page cleaning. The page cleaner threads then clean the LRU queues, marking the writes as *LRU writes*.

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 **OnLine**.

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, since 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

OnLine uses the shared-memory logical-log buffer as temporary storage for records that describe modifications to **OnLine** 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. See Chapter 18, “What Is the Logical Log?,” for a description of logical logging.

Five Events Prompt Flushing of the Logical-Log Buffers

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.
- When a nonlogging database session terminates.
- A checkpoint occurs.
- A page is modified that does not require a before-image in the physical log.

OnLine uses only one of the three logical-log buffers at a time. This buffer is the current logical-log buffer. Before **OnLine** 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, then the third logical-log buffer becomes the current one.

When the Logical-Log Buffer Becomes Full

When a user thread needs to write a page 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.

Once 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 **OnLine** switches to a new logical log, it checks to see if the percentage of used log space is greater than the long-transaction high-water mark, specified by the LTXHWM parameter in the ONCONFIG file. See “LTXHWM” on page 35-25 for a description of the purpose of this parameter and for information on specifying a value for it.

If there is no long-transaction condition, 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 will not fill the logical-log buffer, the thread writes the record, releases latches, and awakens any threads waiting to write to the buffer.

If the write will fill 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. This might cause a waste of some disk space. Typically, many logical-log records are stored on a single page. But 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.

Note, however, that 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

See the SET LOG statement in the *Informix Guide to SQL: Syntax* for a comparison of buffered versus unbuffered logging.

When a Session That Uses Nonlogging Databases or Unbuffered Logging Terminates

Even for nonlogging databases, **OnLine** does log certain activities that alter the database schema, such as the creation of tables or extents. When **OnLine** 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

See “OnLine Checkpoints” on page 14-47 for a detailed description of the events that occur during a checkpoint.

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 OnLine Achieves Data Consistency

OnLine uses the following three procedures to ensure that the data that is destined for disk is actually recorded intact on disk:

- Critical sections
- Checkpoints
- Timestamps

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 **OnLine** code that makes a set of disk modifications that must be performed as a single unit; either all of the modifications must occur or none can occur.

An **OnLine** thread that is in a critical section is holding shared-memory resources. Within the space of the critical section, it is impossible for **OnLine** to 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, **OnLine** takes the two following steps to ensure that all data is returned to the last known point of consistency:

- **OnLine** aborts immediately.
- **OnLine** initiates fast recovery the next time it is initialized.

Fast recovery is the procedure **OnLine** uses to quickly restore the physical and logical consistency of data, up to and including the last record in the logical log. See Chapter 22, “What Is Fast Recovery?,” for a description of fast recovery.

OnLine Checkpoints

The term *checkpoint* refers to the point in **OnLine** 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 **OnLine** is said to be physically consistent.

Five Events 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.
- **OnLine** detects that the next logical-log file to become current contains the most-recent checkpoint record.
- The **OnLine** 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.

One reason an administrator might initiate a checkpoint would be to force a new checkpoint record in the logical log. Forcing a checkpoint would be a step in freeing a logical-log file with status `U-B-L`.

The following section outlines the main events that occur once a user thread raises the checkpoint-requested flag.

Main Events During a Checkpoint

1. **OnLine** prevents user threads from entering critical sections.
2. Page-cleaner thread flushes the physical-log buffer.
3. Page-cleaner threads flush modified pages in the buffer pool to disk. Flushing is performed as a chunk write.
4. Page-cleaner thread writes checkpoint record to logical-log buffer.
5. Physical log on disk is logically emptied (current entries can be overwritten).
6. Logical-log buffer is flushed to current logical-log file on disk.
7. The **main_loop()** thread updates configuration and archive information to reserved pages.

User Threads Cannot Enter a Critical Section

Once the checkpoint requested flag is set, **OnLine** 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.

Page-Cleaner Thread Flushes the Physical-Log Buffer

After all threads have exited from critical sections, the page-cleaner 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 timestamp 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.

Page-Cleaner 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 archive information to the appropriate reserved pages, regardless of whether changes have occurred since the last checkpoint.

When dbspaces, primary chunks, or mirror chunks are added or dropped from **OnLine**, 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 mirror chunks. The `main_loop()` thread writes all checkpoint statistics to the appropriate reserved page in the root dbspace.

Next, the `main_loop` thread looks for logical-log files that can be freed (status U-I) and frees them. Last, the checkpoint-complete record is written to the **OnLine** message log.

Checkpoint Is Critical to Fast Recovery

OnLine generates at least one checkpoint for each span of the logical log to guarantee that it has a checkpoint at which to begin fast recovery.

As fast recovery begins, **OnLine** data is brought to physical consistency as of the last checkpoint by restoring the contents of the physical log.

During the next stage of fast recovery, **OnLine** 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 **OnLine** 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.

Checkpoint Activity During an Archive

Checkpoints that occur during an on-line archive might require slightly more time to complete. The reason is that the archiving procedure forces pages to remain in the physical log until the **onarchive** process or the **ontape** process (depending on which one performs the archive) has had a chance to write the before-image pages to the archive tape. This must be done to ensure that the archive has all timestamped pages needed to complete the archive. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for information on the sequence of events that occur during an archive.

OnLine Timestamps

OnLine uses a timestamp to identify a time when an event occurred relative to other events of the same kind. The timestamp is not a literal time that refers to a specific hour, minute, or second. It is a 4-byte integer that is assigned sequentially. In general, when two timestamps are compared, the one with the lower value is determined to be the older.

Timestamps on Disk Pages

Each disk page has one timestamp in the page header and a second timestamp in the last 4 bytes on the page. The page-header and page-ending timestamps are synchronized after each write, so they should be identical when the page is read from disk. Each read compares the timestamps as a test for data consistency. If the test fails, an error is returned to the **OnLine** 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. See “Structure and Storage of a Dbspace Page” on page 40-30 for a description of the content of a dbspace page.

Timestamps on Blob Pages

In addition to the page-header and page-ending timestamp pair, each disk page that contains a blob also contains one member of a second pair of timestamps. This second pair of timestamps is referred to as the blob timestamp pair. The blob timestamp that appears on the disk page where the blob is stored is paired with a timestamp that is stored with the forward pointer to this blob segment, either in the data row (with the blob descriptor) or with the previous segment of blob data. See “BlobSpace Structure and Storage” on page 40-54 for more information on timestamps on blob pages.

A blob timestamp pair is updated whenever a blob column is updated. When a blob in a data row is updated, the new blob is stored on disk, and the forward pointer stored with the blob descriptor is revised to point to the new location. The blob timestamp in the data row is updated and synchronized with the blob timestamp on the disk page of the new blob.

Blob Timestamps with Dirty Read and Committed Read Isolation Levels

Because retrieving a blob can involve large amounts of data, it might be impossible to retrieve the blob data simultaneously with the rest of the row data. Coordination is needed for blob reads at the Dirty Read or Committed Read level of isolation. Therefore, each read compares the two members of the blob timestamp pair as a test for logical consistency of data. If the two timestamps 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 blob forward pointer read by the **OnLine** user thread is no longer valid.

To understand how a forward pointer stored with a blob descriptor, or with the previous segment of blob 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 **OnLine** user thread is deleting a blob from a data row. During the delete process, another **OnLine** user thread 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 blob is deleted, the space is freed, and a third thread starts to write new blob data in the newly freed space where the first blob was stored. Eventually, when the second **OnLine** user thread starts to read the blob data at the location where the first blob was stored, the thread compares the timestamp from the blob descriptor with the timestamp that precedes the blob data. The timestamps do not match. The blob timestamp on the blobpage is greater than the timestamp 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 since 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 has been fetched. During this lengthy step, it is possible for another program to delete the row, commit the deletion, and for the space on the disk page to be reused. If the space has been reused in the interim, the blob timestamp is greater than the timestamp in the forward pointer. In this case, the comparison indicates the obsolete pointer information and the inconsistency is reported.

Writing Data to a BlobSpace

Blob data that is stored in a dbspace is written to disk pages in the same way as any other data type is written. Blob data that is in a blobSpace (BYTE and TEXT data types) is written to blobSpace pages according to a procedure that differs greatly from the I/O that is performed when data is written to a shared-memory buffer and is then flushed to disk. See “BlobSpace Structure and Storage” on page 40-54 for a description of blobSpaces.

An **OnLine** write operation to a blobSpace differs from a write to a dbspace in several ways.

Blobpages Do Not Pass Through Shared Memory

BlobSpace blobpages store the large amounts of data that have BYTE and TEXT data types. **OnLine** does not create or access blobpages by way of the shared-memory buffer pool. BlobSpace blobpages are not written to either the logical or physical logs.

The reason why blobSpace data is not written to shared memory or to the **OnLine** logs is because the data is potentially very large. If blobSpace data passed through the shared-memory pool, it would dilute the effectiveness of the pool by driving out index pages and data pages. In addition, the many kilobytes of data per blobSpace blob would overwhelm the space allocated for the logical log and the physical log.

Instead, blobpage data is written directly to disk when it is created. Blobpages stored on magnetic media are written to archive and logical-log tapes, but not in the same method as dbspace pages. Blobpages stored on optical media are not written to archive and logical-log tapes due to the reliability of optical media. See “BlobSpace Logging” on page 18-20 for a description of how blobSpaces are logged.

Blobs Are Created Before the Data Row Is Inserted

At the time that the blob data is written to disk, the row itself might not exist yet. During an insert, for example, the blob is transferred before the rest of the row data. After the blob is stored, the data row is created with a 56-byte descriptor that points to the location of the blob. See “Blob Storage and the Blob Descriptor” on page 40-56 a description of how blobs are stored.

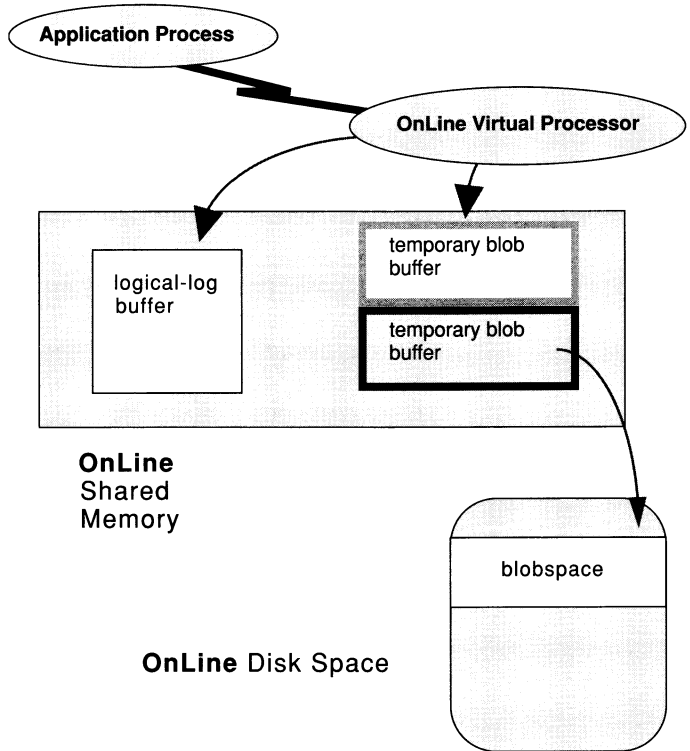
Blobpage Buffers Are Created for the Duration of the Write

To receive blob data from the application process, **OnLine** establishes an open blob for the specific table and row. As part of establishing an open blob, **OnLine** creates a set of blobpage buffers. The set is always composed of two buffers, one buffer for reading and one buffer for writing, each the size of one blobpage. Each user has only one set of blob buffers and, therefore, can access only one blob at a time.

Blob data is transferred from the client application process to **OnLine** in 1-kilobyte segments. **OnLine** begins filling the buffers with the 1-kilobyte pieces and attempts to buffer two blobpages at a time. **OnLine** 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 there is more data to transfer, it adds a forward-pointer to the next page before writing the page to disk. When there is no more data to transfer, **OnLine** writes the last page to disk without a forward pointer.

When the **OnLine** thread begins writing the first blobpage buffer to disk, it attempts to perform the I/O based on the user-defined blobpage size. If, for example, the blobpage size is 32 kilobytes, **OnLine** attempts to read or write blob 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 **OnLine** thread that opened the blob is finished. When the blob has been written to disk, **OnLine** deallocates the blobpage buffers. Figure 14-8 on page 14-54 illustrates the process of creating a blobpage blob.



Writing Data to a BlobSpace:

1. BlobSpace data flows through the client-server connection to temporary buffers in the OnLine shared-memory space, and is written directly to disk. BlobSpace blobpages are allocated and tracked using the free-map page. Links connecting the blobpages and pointers to the next blob segments are created as needed.
2. A record of the operation (insert, update, or delete) is written to the logical-log buffer if the database uses logging.

Figure 14-8 **BlobSpace blobs**

Data is written to a blobSpace without passing through regular shared-memory buffers.

Managing OnLine Shared Memory

Chapter Overview 3

Setting Shared-Memory Configuration Parameters 3

 UNIX Kernel Configuration Parameters 3

 OnLine Shared-Memory Configuration Parameters 6

 Setting Configuration Parameters for the Resident
 Portion of Shared Memory Using
 ON-Monitor 7

 Setting Configuration Parameters for the Resident
 Portion of Shared Memory Using a Text
 Editor 8

 Setting Configuration Parameters for the Virtual
 Portion of Shared Memory Using
 ON-Monitor 9

 Setting Configuration Parameters for the Virtual
 Portion of Shared Memory Using a Text
 Editor 11

 Setting Configuration Parameters for the Shared-
 Memory Performance Options Using
 ON-Monitor 12

 Setting Configuration Parameters for the Shared-
 Memory Performance Options Using a Text
 Editor 13

Reinitializing Shared Memory 14

Turning on or Turning off Residency for Resident Shared
Memory 15

 Turning on or Turning off Residency While OnLine is in
 On-Line Mode 15

 Turning on or Turning off Residency for the Future 15

Adding a Segment to the Virtual Portion of Shared Memory 16
Forcing a Checkpoint 16

Chapter Overview

This chapter tells you how to perform tasks related to managing the use of shared memory with the **INFORMIX-OnLine Dynamic Server**. It assumes you are familiar with the terms and concepts contained in Chapter 14, “OnLine Shared Memory.”

This chapter describes how to perform the following tasks:

- How to set the shared-memory configuration parameters
- How to reinitialize shared memory
- How to turn on or turn off residency for the resident portion of OnLine shared memory
- How to add a segment to the virtual portion of shared memory
- How to force a checkpoint

Setting Shared-Memory Configuration Parameters

You must consider the following two sets of configuration parameters when you configure **OnLine** shared memory:

- UNIX kernel parameters
- **OnLine** shared-memory configuration parameters

The following two sections describe the effects of these two sets of parameters in configuring **OnLine** shared memory.

UNIX Kernel Configuration Parameters

Nine UNIX configuration parameters can affect the use of shared memory by **OnLine**. These parameters are described by function in Figure 15-1 on page 15-4. Parameter names are not provided because names vary among platforms and not all parameters exist on all platforms.

For specific information about your UNIX environment, refer to the machine-specific file, `$INFORMIXDIR/release/ONLINE_6.0`, that is provided with the **OnLine** product.

1	The maximum shared-memory segment size, expressed in bytes or kilobytes
2	The minimum shared-memory segment size, expressed in bytes
3	The maximum number of shared-memory identifiers
4	The shared-memory lower-boundary address
5	The maximum number of attached shared-memory segments per process
6	The maximum amount of shared memory system-wide
7	The maximum number of semaphore identifiers
8	The maximum number of semaphores
9	The maximum number of semaphores per identifier

Figure 15-1 UNIX kernel parameters that can affect OnLine

1 Role of Maximum UNIX Shared-Memory Segment Size

When **OnLine** creates the required shared-memory segments, it attempts to acquire as large an operating-system segment as possible. The first segment size **OnLine** tries to acquire is the size of the portion that it is allocating (resident, virtual, or communications) rounded up to the nearest multiple of eight kilobytes.

OnLine 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 **OnLine** 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 eight kilobytes can be created. Then **OnLine** creates as many additional segments as it requires.

3 Role of Maximum Shared-Memory Identifiers

Shared-memory identifiers affect **OnLine** 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. See “How Virtual Processors Attach to Shared Memory” on page 14-11 for more information about shared-memory identifiers.

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.

4 The Role of the Shared-Memory Lower-Boundary Address

When **OnLine** 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. Your UNIX system might set a parameter, however, 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. See “How Virtual Processors Attach to Shared Memory” on page 14-11 for an illustration of this problem.

5 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:

$$\begin{aligned} \text{Maximum amount of shared memory} = & \\ & (\text{Maximum number of attached shared-memory segments per process}) \\ & \times (\text{Maximum shared-memory segment size}) \end{aligned}$$

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 that which 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.

7 and 9 Semaphore Guidelines

OnLine operation requires one UNIX semaphore for each virtual processor, 1 for each user who connects to **OnLine** through shared memory (ipcshm protocol), 6 for **OnLine** utilities, and 16 for other purposes.

OnLine 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 Using 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 15-2 shows the Shared-Memory screen. The shaded entries set configuration parameters for the resident portion of shared memory.

```

SHARED MEMORY: Make desired changes and press ESC to record changes.
Press Interrupt to abort changes. Press F2 or CTRL-F for field-level help.
                SHARED MEMORY PARAMETERS
Server Number      [  0]          Server Name [odyssey_ol      ]
Server Aliases [oddsoc_ol1,oddsoc_ol2,oddsoc_ol3      ]
Dbospace Temp     [          ]
Deadlock Timeout  [  0] Secs   Number of Page Cleaners [  1]
Forced Residency  [Y]          Stack Size (Kbytes) [ 32]
Non Res. SegSize (Kbytes) [ 4000]

Physical Log Buffer Size [  32] Kbytes
Logical Log Buffer Size [  32] Kbytes
Max # of Logical Logs [  14]
Max # of Transactions [  20]
Max # of Userthreads [  20]
Max # of Locks        [ 2000]
Max # of Buffers      [  80]
Max # of Chunks       [  10]
Max # of Open Tblspaces [ 200]
Max # of Dbspaces     [  10]
=====
Shared memory size [  546] Kbytes      Page Size [  2] Kbytes

Enter a unique value to be associated with this version of INFORMIX-OnLine.

```

Figure 15-2 ON-Monitor Shared-Memory screen

Note: 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 15-3 shows only the Shared-Memory screen entries that affect the configuration of the resident portion of shared memory. For each entry, it shows within brackets ([]), the name of the associated parameter in the ONCONFIG file.

```

SHARED MEMORY: Make desired changes and press ESC to record changes.
Press Interrupt to abort changes. Press F2 or CTRL-F for field-level help.
                SHARED MEMORY PARAMETERS
Server Number      [SERVER_NUM]

                Number of Page Cleaners[CLEANERS]
Forced Residency   [Y]

Physical Log Buffer Size [ PHYSBUFF] Kbytes
Logical Log Buffer Size [ LOGBUFF] Kbytes
Max # of Logical Logs  [ LOGFILES]
Max # of Transactions  [TRANSACTIONS]
Max # of Userthreads   [USERTHREADS]
Max # of Locks         [ LOCKS]
Max # of Buffers       [ BUFFERS]      Add SegSize (Kbytes) [SHMADD]
Max # of Chunks        [ CHUNKS]      Total Memory (Kbytes) [SHMTOTAL]
Max # of Open Tblspaces [TBLSPACES]
Max # of Dbspaces      [DBSPACES]

=====
Shared memory size   [      546] Kbytes      Page Size [  2] Kbytes

Enter a unique value to be associated with this version of INFORMIX-OnLine.
    
```

Figure 15-3 Partial view of ON-Monitor Shared-Memory screen showing the ONCONFIG parameter for each of the shared-memory entries

See Figure 15-4 on page 15-9 for more information on the ONCONFIG parameters that are associated with the resident portion of shared memory.

Setting Configuration Parameters for the Resident Portion of Shared Memory Using 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.

Figure 15-4 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 35, “OnLine Configuration Parameters.”

Parameter	Purpose	Page
BUFFERS	Specifies the maximum number of shared-memory buffers	page 35-8
CHUNKS	Specifies the maximum number of chunks	page 35-9
CLEANERS	Specifies the number of page-cleaner threads that OnLine is to run	page 35-11
DBSPACES	Specifies the maximum number of dbspaces	page 35-13
LOCKS	Specifies the maximum number of locks for database objects—for example, rows, key values, pages, and tables	page 35-19
LOGBUFF	Specifies the size of the logical-log buffers	page 35-20
LOGFILES	Specifies the number of logical-log files OnLine is to create during disk initialization	page 35-20
PHYSBUFF	Specifies the size of the physical-log buffers	page 35-32
RESIDENT	Specifies residency for the resident portion of OnLine shared memory	page 35-35
SERVERNUM	Specifies a unique identification number for OnLine on the local host computer	page 35-37
SHMADD	Specifies the size of dynamically-added shared-memory segments	page 35-37
SHMTOTAL	Specifies the total amount of memory to be used by OnLine	page 35-38
TBLSPACES	Specifies the maximum number of active tblspaces	page 35-44
TRANSACTIONS	Specifies the maximum number of concurrent transactions that OnLine can handle	page 35-45
USERTHREADS	Specifies the maximum number of user threads that OnLine can run	page 35-46

Figure 15-4 ONCONFIG parameters for configuring the resident portion of shared memory

Setting Configuration Parameters for the Virtual Portion of Shared Memory Using 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 15-8 shows the Shared-Memory screen. The shaded entries set configuration parameters for the virtual portion of shared memory.

```

SHARED MEMORY: Make desired changes and press ESC to record changes.
Press Interrupt to abort changes. Press F2 or CTRL-F for field-level help.
                SHARED MEMORY PARAMETERS
Server Number      [ 2 ]          Server Name [lashley_ol ]
Server Aliases [cole,davison,stackhouse ]
Dbospace Temp [ ]
Deadlock Timeout  [ 60 ] Secs  Number of Page Cleaners [ 1 ]
Forced Residency  [ N ]          Stack Size (Kbytes) [ 32 ]
Non Res. SegSize (Kbytes) [ 4000 ]

Physical Log Buffer Size [ 32 ] Kbytes
Logical Log Buffer Size [ 32 ] Kbytes
Max # of Logical Logs [ 14 ]          Transaction Timeout [ 300 ]
Max # of Transactions [ 20 ]          Long TX HWM [ 80 ]
Max # of Userthreads [ 20 ]          Long TX HWM Exclusive [ 90 ]
Max # of Locks [ 2000 ]          Index Page Fill Factor [ 90 ]
Max # of Buffers [ 80 ]
Max # of Chunks [ 10 ]
Max # of Open Tblspaces [ 200 ]
Max # of Dbspaces [ 10 ]

=====
Shared memory size [ 528 ] Kbytes          Page Size [ 2 ] Kbytes
Enter a unique value to be associated with this version of INFORMIX-OnLine.
    
```

Figure 15-5 The ON-Monitor Shared Memory screen; shaded entries set virtual shared-memory configuration parameters

Figure 15-6 shows only the Shared Memory screen entries that affect the configuration of the virtual portion of shared memory. For each entry, it shows, within brackets ([]), the name of the associated parameter in the ONCONFIG file.

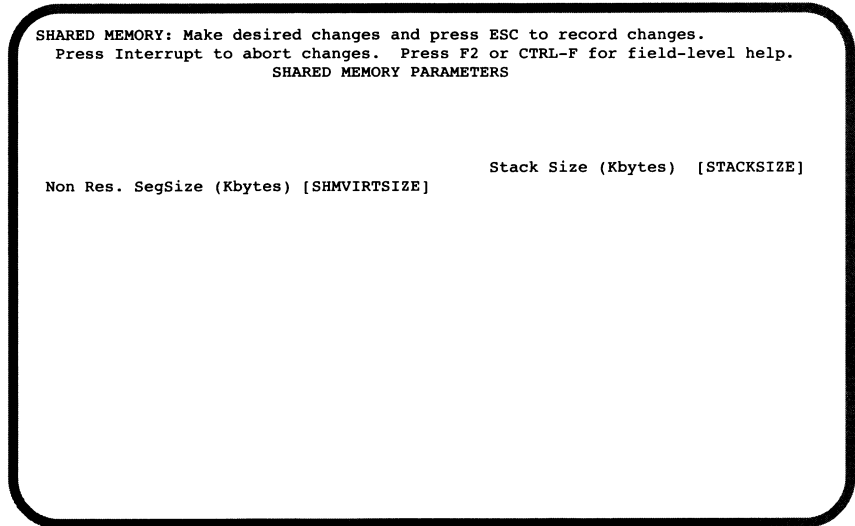


Figure 15-6 *Partial view of the ON-Monitor Shared Memory screen showing the ONCONFIG parameters for each of the virtual shared-memory entries*

See Figure 15-7 on page 15-11 for more information on the ONCONFIG parameters that affect the configuration of the virtual portion of shared memory.

Setting Configuration Parameters for the Virtual Portion of Shared Memory Using 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 using 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 15-7 lists the ONCONFIG parameters that you use to configure the virtual portion of shared memory

Parameter	Purpose	Page
SHMVIRTSIZE	Specifies the initial size of the virtual portion of shared memory	page 35-39
STACKSIZE	Specifies the stack size for OnLine user threads	page 35-40

Figure 15-7 *ONCONFIG parameters for configuring the virtual portion of shared memory*

Setting Configuration Parameters for the Shared-Memory Performance Options Using 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 15-8 shows the perFormance screen. The shaded entries set the configuration parameters for the shared-memory performance options.

```

PERFORMANCE: Make desired changes and press ESC to record changes.
Press Interrupt to abort changes. Press F2 or CTRL-F for field-level help.
PERFORMANCE TUNING PARAMETERS

CPU VPs [ 3]
AIO VPs [ 1]

      Protocol Threads  Users  VP-class
ipcshm [N] [ ] [ ] [ ]
tlitcp [N] [ ] [ ] [ ]
tlispx [N] [ ] [ ] [ ]
soctcp [N] [ ] [ ] [ ]

Multiprocessor Machine [N]
Disable Priority Aging [N]
Num Procs to Affinity [ 0]
Proc num to start with [ 0]

Single CPU VP [N]
Mutex Wait Lists [N]
Use OS Time [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]

Enter the number of CPU virtual processors.
    
```

Figure 15-8 ON-Monitor perFormance screen

Figure 15-9 shows only the perFormance screen entries for setting the shared-memory performance options. For each entry, it shows, within brackets ([]), the name of the associated parameter in the ONCONFIG file.

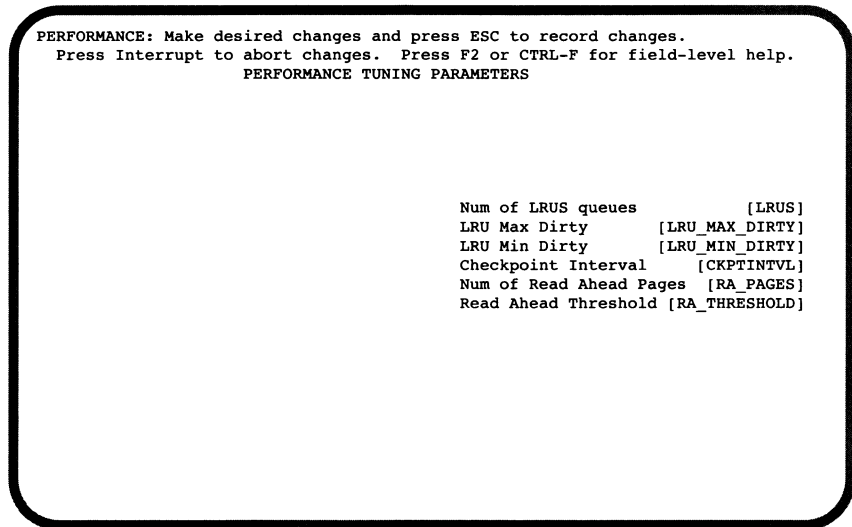


Figure 15-9 *Partial view of ON-Monitor perFormance screen showing the ONCONFIG parameters for the shared-memory performance options*

See Figure 15-10 on page 15-14 for more information on the ONCONFIG parameters that set shared-memory performance options.

Setting Configuration Parameters for the Shared-Memory Performance Options Using 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.

Figure 15-10 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 35, “OnLine Configuration Parameters.”

Parameter	Purpose	Page
CKPTINTVL	Specifies the maximum number of seconds that can elapse before OnLine checks to see if a checkpoint is needed	page 35-10
LRU_MAX_DIRTY	Specifies the percentage of modified pages in the LRU queues that flags page cleaning to start	page 35-22
LRU_MIN_DIRTY	Specifies the percentage of modified pages in the LRU queues that flags page cleaning to stop	page 35-23
LRUS	Specifies the number of LRU queues for the shared-memory buffer pool	page 35-22
RA_PAGES	Specifies the number of disk pages that OnLine should attempt to read ahead when doing sequential scans of data or index records	page 35-34
RA_THRESHOLD	Specifies the number of memory pages which, after they are read, cause OnLine to read-ahead on disk	page 35-34

Figure 15-10 *ONCONFIG parameters for setting shared-memory performance options*

Reinitializing Shared Memory

OnLine re-initializes shared memory when you take **OnLine** 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, **OnLine** must first be brought off line. See Chapter 8, “Managing Modes,” for information on how to take **OnLine** from on-line mode to off-line.

After **OnLine** is off-line, you need to bring it to quiescent mode or on-line mode to reinitialize shared memory. See “From Off-Line to Quiescent” on page 8-3 and “From Off-Line to On-Line” on page 8-4.

Turning on or Turning off Residency for Resident Shared Memory

You can turn on or turn off residency for the resident portion of shared memory in either of the following two ways.

- You can use the **onmode** utility to immediately reverse the state of shared-memory residency while **OnLine** is in on-line mode.
- You can change the **RESIDENT** parameter in the **ONCONFIG** file to turn shared-memory residency on or off for the next time you initialize **OnLine** shared memory.

See “The Resident Portion of OnLine Shared Memory” on page 14-15 for a description of the resident portion of shared memory.

Turning on or Turning off Residency While OnLine is in On-Line Mode

You can turn on or turn off residency while **OnLine** is in on-line mode by using the **onmode** utility. You must be **root** or user **informix** to do this.

To immediately turn on residency for the resident portion of shared memory, execute the following command:

```
% onmode -r
```

To immediately turn off residency for the resident portion of shared memory, execute the following command:

```
% onmode -n
```

This change does 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 on or Turning off Residency for the Future

You can turn on or turn off residency for the next time **OnLine** shared memory is initialized by changing the value of the **RESIDENT** parameter in the **ONCONFIG** file. You can change the **RESIDENT** parameter by using either **ON-Monitor** or a text editor. The following sections describe how to change the **RESIDENT** parameter using both of these methods.

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 **OnLine** 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

Occasionally **OnLine** cannot free a logical-log file even though it is backed up to tape and all transactions within it are closed. This situation arises when the logical-log file contains the most-recent checkpoint in the logical logs. This log file must maintain a backed-up status until a new checkpoint record is written to the current logical-log file. **OnLine** processing stops until the new checkpoint record is written to the current logical-log file. See “What Happens If the Next Logical-Log File Is Not Free?” on page 18-11 for more information on this condition.

As user **informix**, you can force a checkpoint using ON-Monitor 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
```



Logging and Log Administration



What Is Logging?

Chapter Overview 3

Which OnLine Processes Require Logging? 3

What OnLine Activity Is Logged? 5

Activity That Is Always Logged 6

Activity Logged for Databases with Transaction
Logging 6

Are Blobs Logged? 7

What Is Transaction Logging? 7

The Database Logging Status 8

Unbuffered Transaction Logging 8

Buffered Transaction Logging 9

ANSI-Compliant Transaction Logging 9

If Some Databases Use Buffered and Some Use
Unbuffered Logging 9

When to Use or not Use Transaction Logging 9

When to Buffer or not Buffer Transaction Logging 10

Who Can Set or Change Logging Status 10

Chapter Overview

This chapter describes the functionality of **INFORMIX-OnLine Dynamic Server** logging. First logging is described with respect to **OnLine** functionality. The following questions are addressed:

- Which **OnLine** features require logging?
- What **OnLine** activity is logged?

Next, logging is described with respect to databases. You specify whether or not a database uses *transaction logging* and, if it does, what log buffering mechanism it uses. The following questions are addressed:

- 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 OnLine Processes Require Logging?

As **OnLine** 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 **OnLine** makes no further use of the log records. But when **OnLine** needs to roll back a transaction, for example, or to execute a fast recovery after a system failure, the log records are critical. The log records are at the heart of **OnLine** data-recovery mechanisms.

OnLine stores the log records in a *logical log*. The logical log is made up of *logical-log files* that **OnLine** manages on disk until they have been safely transferred off-line (*backed up*). The **OnLine** administrator keeps the off-line log records (in the backed up logical-log files) until they are needed during a data restore, or the administrator decides they are no longer needed for a restore. Logical-log administration topics are explained in Chapter 18, “What Is the Logical Log?”

The records written to the logical log are necessary for various functions that **OnLine** performs to recover data and ensure data consistency. **OnLine** requires logical-log records to perform the following processes:

- **Fast recovery**

If **OnLine** shuts down in an uncontrolled manner, **OnLine** uses the log records to recover all transactions that occurred since the most-recent checkpoint—when all the data in shared memory and all the data on disk were the same (also known as *physically consistent*)—and to roll back any uncommitted transactions. The log records are used in the second phase of fast recovery when **OnLine** returns the entire database server to a state of logical consistency up to the point of the most-recent logical-log record. (See “Details of Fast Recovery” on page 22-5 for more information.)
- **Transactions roll back**

If a database has transaction logging turned on (see “What Is Transaction Logging?” on page 16-7) and a transaction must be rolled back, **OnLine** 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 **OnLine** archive tapes to re-create the **OnLine** system up to the point of the most-recently backed up logical-log record. After the archive tapes have been restored, **OnLine** essentially reimplements all the logged activity since the last archive using the log records.
- **Deferred checking**

If a transaction uses the SET CONSTRAINTS statement to set checking to DEFERRED, constraints are not checked until the transaction is committed. If a constraint error occurs while the transaction is being committed, logical-log records from the transaction are used 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**

The logical-log records of a distributed transaction are kept on disk by each of the **OnLine** database servers involved. This ensures data integrity and consistency, even if a failure occurs on one of the **OnLine** database servers performing the transaction. See “Two-Phase Commit and Logical Log Records” on page 32-28 for more information.

- 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. See “How Does Data Replication Work?” on page 25-8 for a more detailed discussion of how **OnLine** data replication uses logical-log records.

What OnLine Activity Is Logged?

OnLine does not generate log records for every operation because it does not need a record of every action. **OnLine** only needs log records to perform the functions listed under “Which OnLine Processes Require Logging?” on page 16-3. Also, the space required to store a record of everything the database server did would quickly be overwhelming.

The logical-log records themselves are of variable length. This increases the number of log records that can be written to a page in the logical-log buffer. However, often the logical-log buffer is flushed before the page is full.

Two types of logged activity are possible in **OnLine**:

- Activity that is always logged
- Activity that is only logged for databases using transaction logging

These different types are explained in the following sections. For more information on the format of logical-log records, refer to Chapter 39, “Interpreting Logical-Log Records.”

Activity That Is Always Logged

Some database operations always generate logical-log records, even if none of the databases on the database server use transaction logging. (See “What Is Transaction Logging?” on page 16-7.) These operations are as follows:

- SQL data definition statements for all databases:

ALTER INDEX	DATABASE
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
CREATE VIEW	

- Archive events
- Checkpoint events
- Administrative changes to the **OnLine** configuration
This category includes changes to the number and location of chunks, dbspaces, and blobspaces.
- Allocation of new extents to tables that are growing in size
- A change to the logging status of a database

Activity Logged for Databases with Transaction Logging

If a database uses transaction logging, all SQL data manipulation statements (DML)—except SELECT—against that database generate one or more log records. Those statements are as follows:

- DELETE
- INSERT
- LOAD
- SELECT INTO TEMP
- UNLOAD
- UPDATE

If these statements are rolled back, the rollback also generates log records.

Are Blobs Logged?

Blob data is potentially too voluminous to be part of a logical-log record. If blob data were always included, the many kilobytes of data per blob could overwhelm the space allocated for the logical log. However, not all blobs are that large, and not every blob would overwhelm the logical log.

OnLine assumes that you have designed your databases so that smaller blobs are stored in dbspaces, and larger blobs are stored in blobspaces. (See the *Informix Guide to SQL: Tutorial* for a discussion of locating blob data). Based on this assumption:

- **OnLine** includes blob data in log records for blobs stored in dbspaces.
- **OnLine** does not include blob data in log records for blobs stored in blobspaces.

OnLine still needs access to the blob data for blobspace blobs to fulfill the goals of logging, explained in “Which OnLine Processes Require Logging?” on page 16-3. Consider the following question: How can a blobspace operation 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. This mechanism is described under “Blobspace Logging” on page 18-20.

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 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) once the database is created. For information on changing the database logging status, refer to “Who Can Set or Change Logging Status” on page 16-10, and to Chapter 17, “Managing Database Logging Status.”

Even if transaction logging is turned off for all databases in an **OnLine** database server, some **OnLine** events are always logged (those listed in “Activity That Is Always Logged” on page 16-6).

The Database Logging Status

Every database managed by **OnLine** has a logging status. The logging status indicates whether the database uses transaction logging and, if so, which log-buffering mechanism the database employs. You can find out the transaction logging status of a database using **OnLine** utilities, as explained in “Monitoring Databases” on page 29-36. 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 OnLine Uses Shared Memory” on page 14-6, information that **OnLine** manages passes through shared memory to disk. Logical-log records are no exception. Before log records are written to the logical log, which is on disk, they must pass through shared memory. They do this through the logical-log buffers, explained in “Flushing the Logical-Log Buffer” on page 14-44.

In one sense, all **OnLine** logging is buffered because all log records pass through the logical-log buffer in shared memory before they are written to the logical log on disk. However, the point at which the logical-log buffer is flushed is different for buffered transaction logging and unbuffered transaction logging.

Unbuffered Transaction Logging

If transactions are made against a database that uses unbuffered logging, the records in the logical-log buffer are guaranteed to be written to disk before the COMMIT statement (and before the PREPARE statement for distributed transactions) returns to the application. The records are flushed as soon as any transaction in the buffer is committed (that is a commit record is written to the logical-log buffer).

When the buffer is flushed only the used pages are written to disk. This includes 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 **OnLine** 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.
- The buffer is flushed by a commit on a database with unbuffered logging.
- 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.

If Some Databases Use Buffered and Some Use Unbuffered Logging

Because all databases use the same logical log and the same three logical-log buffers, it is possible (and probable) for transactions against databases with different log-buffering status to be writing to the same logical-log buffer. In that case—if there are transactions against databases with buffered logging *and* transactions against databases with unbuffered logging—the buffer is flushed *either* when it is full, *or* when transactions against the database(s) with unbuffered logging complete.

When to Use or not Use Transaction Logging

You must use transaction logging with a database to take advantage of any of the features listed in “Which OnLine Processes Require Logging?” on page 16-3.

If you are satisfied with your recovery source, you can decide not to use transaction logging for a database to reduce the amount of **OnLine** 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 logging is reinitiated, which must wait for a level-0 archive.

If you are operating in 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). So, if one of the databases in a distributed query uses transaction logging, the others must also.

When to Buffer or not 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 and cannot have their buffering status changed.

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. The cost of this is that the logical-log buffer is flushed to disk more frequently and contains many more partially full pages, so it fills the logical log faster than buffered logging.

If you use buffered logging and a failure occurs, you cannot expect **OnLine** 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 is slightly improved. 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 in order 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 establishes the logging status for that database when it is created with the `CREATE DATABASE` statement. If the creator of a database does not specify a logging status with the `CREATE DATABASE` statement, the database is created without logging. See the *Informix Guide to SQL: Syntax* for more information on the `CREATE DATABASE` statement.

Only the **OnLine** administrator can change the logging status. This topic is described in Chapter 17, "Managing Database Logging Status." Ordinary end users cannot change the database logging status. End users *can* switch from unbuffered transaction logging 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. See the *Informix Guide to SQL: Syntax* for more information on the SET LOG statement.

Managing Database Logging Status

Chapter Overview	3
About Changing Logging Status	3
Modifying Database Logging Status Using ON-Archive	5
Turning on Transaction Logging Using ON-Archive	5
Canceling a Logging Operation Using ON-Archive	5
Ending Logging Using ON-Archive	6
Changing Buffering Status Using ON-Archive	6
Making a Database ANSI-Compliant Using ON-Archive	6
Modifying Database Logging Status Using ontape	6
Turning on Transaction Logging Using ontape	7
Ending Logging Using ontape	7
Changing Buffering Status Using ontape	7
Making a Database ANSI-Compliant Using ontape	8
Modifying Database Logging Status Using ON-Monitor	8

Chapter Overview

This chapter provides instructions on changing the database logging status for databases managed by **INFORMIX-OnLine Dynamic Server**. As an **OnLine** 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 16, “What Is Logging?”

To find out the current logging status of a database, see “Monitoring Databases” on page 29-36.

About Changing Logging Status

Figure 17-1 on page 17-4 shows which database logging status transitions the **OnLine** administrator can perform and whether they take place immediately or require a level-0 archive.

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 archive is performed on all the dbspaces and blobspaces that contain data in the database. You should use the same archiving tool (**ON-Archive** or **ontape**) to add logging and create the archive.

You cannot make any changes to the logging status of ANSI-compliant databases.

Converting from:	Converting to:			
	No logging	Unbuffered logging	Buffered logging	ANSI-compliant
No logging	Not applicable	Level-0 archive (of affected dbspaces)	Level-0 archive (of affected dbspaces)	Level-0 archive (of affected dbspaces)
Unbuffered logging	Immediate	Not applicable	Immediate	Immediate
Buffered logging	Immediate	Immediate	Not applicable	Immediate
ANSI-compliant	Illegal	Illegal	Illegal	Not applicable

Figure 17-1 Logging status transitions

Here are some general points about changing the database logging status:

- To make any change in the logging status of a database, no users can access the database. Once you start to make the change, you have an exclusive lock on the database, preventing other users from accessing the database.
- A database remains locked to users until the logging mode change is complete. For most changes this is immediate, but if you add logging to a database that formerly did not have logging, the change is not complete until the next level-0 archive of all the dbspaces containing data for the database.
- If you are using ON-Archive, **OnLine** must be in on-line mode to make the changes. For **ontape** or ON-Monitor, **OnLine** can be in either on-line or quiescent mode.
- If a failure occurs while you are making a logging-mode change, you should check the flags for the database (see “Monitoring Databases” on page 29-36) once the database server (or dbspace) is restored.
- Once you choose either buffered or unbuffered logging, it is possible to change from either logging status to the other, within an application using the SQL statement SET LOG. This change lasts for the duration of the session.

Modifying Database Logging Status Using ON-Archive

The command you use to modify database logging status with ON-Archive is `MODIFY/DBLOGGING`. Reference information appears in the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*.

Turning on Transaction Logging Using ON-Archive

Before making this change, read “About Changing Logging Status” on page 17-3.

To add buffered logging to a database called `stores6` using ON-Archive, use the following command:

```
Onarchive> MODIFY/DBLOGGING=stores6/MODE=BUFFERED
```

You can also use the same type of request to add unbuffered logging by using `UNBUFFERED` as a parameter to the `MODE` qualifier.

Canceling a Logging Operation Using ON-Archive

After you turn on logging for a database using ON-Archive, you can turn logging off again (and unlock the database) before the next level-0 archive 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 archived, issue the following query to the `sys-master` database:

```
SELECT name FROM sysdatabases WHERE flags > 255
```

To turn logging off for a database called `stores6` using ON-Archive (after it has been turned on, but before the change is completed by a level-0 archive), use the following command:

```
Onarchive> MODIFY/DBLOGGING=stores6/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 Using ON-Archive

Before making this change, read “About Changing Logging Status” on page 17-3.

To end logging for a database called **stores6** using ON-Archive, use the following command:

```
Onarchive> MODIFY/DBLOGGING=stores6/MODE=NOLOGGING
```

Changing Buffering Status Using ON-Archive

Before making this change, read “About Changing Logging Status” on page 17-3.

To change the buffering status for a database called **stores6** using transaction logging using ON-Archive, use one of the following commands, depending on whether you wish the database to have buffered logging or not:

```
Onarchive> MODIFY/DBLOGGING=stores6/MODE=BUFFERED
```

```
Onarchive> MODIFY/DBLOGGING=stores6/MODE=UNBUFFERED
```

Making a Database ANSI-Compliant Using ON-Archive

Before making this change, read “About Changing Logging Status” on page 17-3. Remember that once you change the logging status to ANSI-compliant, you cannot easily change it again. You have to unload and reload the data.

To make a database called **stores6** ANSI-compliant using ON-Archive, use the following command:

```
Onarchive> MODIFY/DBLOGGING=stores6/MODE=ANSI
```

Modifying Database Logging Status Using *ontape*

You can use **ontape** to change the logging status of a database, unless you are adding logging to a database and generally use ON-Archive to create archives. In that situation, you should use ON-Archive to turn on transaction logging. See “Turning on Transaction Logging Using ON-Archive” on page 17-5.

Reference information for **ontape** is in “Change Database Logging Status” on page 37-73.

Turning on Transaction Logging Using *ontape*

Before making this change, read “About Changing Logging Status” on page 17-3.

You add logging to a database using **ontape** at the same time you create a level-0 archive.

For example, to add buffered logging to a database called **stores6** using **ontape**, execute the following command:

```
% ontape -s -B stores6
```

To add unbuffered logging to a database called **stores6** using **ontape**, execute the following command:

```
% ontape -s -U stores6
```

In addition to turning on transaction logging, these commands create full-system archives. When **ontape** prompts you for an archive level, you should specify a level-0 archive.

Note that with **ontape** you must perform a level-0 archive of all dbspaces. The ON-Archive utility permits greater archiving granularity, so you can restrict the archive to only those dbspaces which contain the database data.

Ending Logging Using *ontape*

Before making this change, read “About Changing Logging Status” on page 17-3.

To end logging for a database called **stores6** using **ontape**, execute the following command:

```
% ontape -N stores6
```

Changing Buffering Status Using *ontape*

Before making this change, read “About Changing Logging Status” on page 17-3.

To change the buffering status from buffered to unbuffered logging on a database called **stores6** using **ontape** without creating an archive, execute the following command:

```
% ontape -U stores6
```

To change the buffering status from unbuffered to buffered logging on a database called **stores6** using **ontape** without creating an archive, execute the following command:

```
% ontape -B stores6
```

Making a Database ANSI-Compliant Using *ontape*

Before making this change, read “About Changing Logging Status” on page 17-3. Remember that once you change the logging status to ANSI-compliant, you cannot easily change it again. You have to unload and reload the data.

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 **stores6**, which already uses transaction logging (either unbuffered or buffered), into an ANSI-compliant database using **ontape**, execute the following command:

```
% ontape -A stores6
```

To make a database called **stores6**, which does not already use transaction logging, into an ANSI-compliant database using **ontape**, execute the following command:

```
% ontape -s -A stores6
```

In addition to making a database ANSI-compliant, this command also creates an archive at the same time. You should specify a level-0 archive when prompted for a level.

Modifying Database Logging Status Using ON-Monitor

Before you make any changes, be sure to read “About Changing Logging Status” on page 17-3.

You can use ON-Monitor to make any logging status changes that can occur immediately. If you want to add logging to (or make ANSI-compliant) a database that does not use logging, you cannot use ON-Monitor; you must use ON-Archive or **ontape** to add logging.

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?

Chapter Overview	3
What Is the Logical Log?	3
What Is a Logical-Log File?	4
How Big Should the Logical Log Be?	5
Performance Considerations	5
Long-Transaction Consideration	6
Logical-Log Size Guidelines	6
Determining the Size of the Logical Log	7
What Should Be the Size and Number of Logical-Log Files?	7
Where Should Logical-Log Files Be Located?	8
How Are Logical-Log Files Identified?	8
What Are the Status Flags of Logical-Log Files?	9
Why Do Logical-Log Files Need to Be Backed Up?	10
When Are Logical-Log Files Freed?	11
When Does OnLine Attempt to Free a Log File?	11
What Happens If the Next Logical-Log File Is Not Free?	11
Avoiding Long Transactions	12
Factors That Influence the Rate at Which Logical-Log Files Fill	13
Factors That Prevent Closure of Transactions	14
Setting High-Water Marks	14

What Are the Logical-Log Administration Tasks Required for Blobspaces?	15
Switching Logical-Log Files to Activate Blobspaces	15
Switching Logical-Log Files to Activate New Blobospace Chunks	16
Backing Up Logical-Log Files to Free Blobspages	16
Why Do You Have to Back-Up Logical-Log Files to Free Blobspages?	16
What Is the Logging Process?	18
Dbospace Logging	18
Read Page into Shared-Memory Buffer Pool	19
Copy the Page Buffer into the Physical-Log Buffer	19
Read Data into Buffer; Create Logical-Log Record	19
Flush Physical Log Buffer to the Physical Log	19
Flush Page Buffer	20
Flush Logical-Log Buffer	20
Blobospace Logging	20

Chapter Overview

As an **INFORMIX-OnLine Dynamic Server** administrator you have certain responsibilities that deal with configuring and running the logical log. These responsibilities include the following tasks:

- Allocating an appropriate amount of disk space for the logical log
- Choosing how many logical-log files are stored in that disk space
- 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, background information on the **OnLine** logging process is given here.

For more information on backing up logical-log files—a critical part of managing logical-log files—refer to the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*.

For information on how to perform other logical-log tasks refer to Chapter 19, “Managing Logical-Log Files.”

What Is the Logical Log?

OnLine keeps a history of database and database server changes since the time of the last archive by generating and storing log records. The log records are stored 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 **OnLine** archive tapes plus **OnLine** logical-log files contain a complete copy of your **OnLine** data.

All the databases managed by a single **OnLine** database server store their log records in the same logical log, regardless of whether they use transaction logging or not, or whether their transaction logging is buffered or not. For information on transaction logging, see Chapter 16, “What Is Logging?”

Most end users should not be concerned with the logical log and logical-log files. They might be concerned with the buffering status of a database during their transactions, or even occasionally with the transaction-logging status of a database, as explained in “Who Can Set or Change Logging Status” on page 16-10.

Most of the administration of the logical log concerns the management of individual logical-log files, but there is one administrative task related 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 **OnLine**; 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 **OnLine** administrator needs to be concerned with the logical-log files that make up the logical log because if they are not managed properly **OnLine** can suspend processing and, in the worst case, shut down.

It is the **OnLine** administrator’s responsibility to choose an appropriate number, size, and physical location for logical-log files. This topic is discussed in the following sections:

- “What Should Be the Size and Number of Logical-Log Files?” on page 18-7
- “Where Should Logical-Log Files Be Located?” on page 18-8

It is also the **OnLine** administrator’s responsibility to ensure that the next logical-log file is always backed up and free. This topic is discussed in the following sections:

- “Why Do Logical-Log Files Need to Be Backed Up?” on page 18-10
- “When Are Logical-Log Files Freed?” on page 18-11

There are also some logical-log file administration tasks related to effectively managing blobspaces. These topics are discussed in the following section:

- “What Are the Logical-Log Administration Tasks Required for Blobspaces?” on page 18-15

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 is necessary, space will be 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.

- 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 18-10.) The backup process can hinder transaction processing involving data located on the same disk as the logical-log files. If there is enough logical-log disk space, however, you can wait for periods of low user activity before backing 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 containing the last checkpoint, **OnLine** must write a new checkpoint record to the current logical-log file. (See “When Are Logical-Log Files Freed?” on page 18-11.) 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 also determine the checkpoint frequency (such as the physical log size), 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 having a long-transaction situation if logical-log disk space is insufficient. For more information on the long-transaction situation, refer to “Avoiding Long Transactions” on page 18-12.

Logical-Log Size Guidelines

It is difficult to predict how much logical-log space is sufficient for your system until the system is actually in use. The *minimum* value you should configure initially is given by the following expression:

$$(3 \text{ Log Files}) * (\text{Number of Userthreads}) * (2 \text{ Log Pages}) * (\text{Page Size})$$

Two log pages is the maximum that a userthread can log during a critical section. Three log files is the minimum number of log files allowed. So, for example, if the number of userthreads is 100 and the page size is 2 kilobytes, the minimum value of the logical-log size that you should configure is 600 kilobytes.

The value yielded by this expression is the minimum logical-log disk space required to ensure that a checkpoint record can be written in a logical-log span. In other words, even if all threads are in critical sections of code when the checkpoint occurs, there is enough logical-log space for all the threads to complete their transactions before the checkpoint record is written.

You can increase the value yielded by the preceding expression as necessary. You can increase the amount of space devoted to the logical log in several ways. The easiest way is to add another logical-log file, as explained in “Adding a Logical-Log File” on page 19-3.

Determining the Size of the Logical Log

The total space allocated to the logical-log files is equal to the number of logical-log files multiplied by the size of each log. The number of logical-log files is given by the LOGFILES parameter. The size of the logical-log files is given by the LOGSIZE parameter. The **OnLine** administrator sets both of these configuration parameters. Both LOGFILES and LOGSIZE are stored in the ONCONFIG file.

If you have added logical-log files that are not of the size specified by LOGSIZE, the (LOGFILES * LOGSIZE) expression does not tell you how large your logical log is. 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 29-37.

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, you need to ensure that logical-log files are small enough to be backed up in a timely fashion.
- Smaller log files means smaller granularity of recovery because you potentially lose the last unbacked-up logical-log file if the disk containing the logical-log files goes down.

When you think about the *number* of logical-log files, consider these points:

- You must always have at least three logical-log files.
- You should create enough logical-log files so that you can switch log files if needed without running out of free logical-log files.
- The number of logical-log files affects the frequency of logical-log backups and, consequently, the rate at which blobpages can be reclaimed. (See “Backing Up Logical-Log Files to Free Blobpages” on page 18-16.)
- The number of logical-log files cannot exceed the value of the ONCONFIG parameter LOGSMAX.

Where Should Logical-Log Files Be Located?

When **OnLine** disk space is initialized, the logical-log files and the physical log are placed in the root dbspace. You have no control over this. To improve performance (specifically, to reduce the number of writes to the root dbspace and minimize contention), move the logical-log files out of the root dbspace to a dbspace on a disk that is not shared by active tables or the physical log. (See “Moving a Logical-Log File to Another Dbspace” on page 19-6.)

You can improve performance further by separating the logical-log files into two groups and storing them on two separate disks (neither of which contain data). For example, if you have six logical-log files, you would locate files 1, 3, and 5 on disk 1, and files 2, 4, and 6 on disk 2. Performance is improved 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 the dbspace to which you are moving the log files is not already mirrored, plan to start mirroring 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 **OnLine** disk space is initialized. When the current logical-log file becomes full, **OnLine** 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 have configured six logical-log files, these files have logid numbers one through six. As logical-log files are backed up and freed (see “Why Do Logical-Log Files Need to Be Backed Up?” on page 18-10), **OnLine** reuses the disk space for the logical-log files; however, **OnLine** continues to increment the unique id numbers by one.

Figure 18-1 illustrates the relationship between the logid numbers and the unique id numbers.

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

Figure 18-1 Logical-log file numbering sequence

For information on how to display the unique id and logid numbers of a logical-log file, refer to “Monitoring Logical-Log Files” on page 29-37.

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 archive 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 **OnLine** for recovery (rollback of a transaction or finding the last checkpoint record).
- Backed-Up (B) A log file has a *backed up* status after the log file has been backed up.

Current (C)	A log file has a <i>current</i> status if OnLine 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 OnLine writes a new checkpoint record to a different logical-log file.

Figure 18-2 shows the possible log status flag combinations.

Status Flag	Status of Logical-Log File
A-----	Log has been added since last level-0 archive. 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.

Figure 18-2 Summary of possible logical-log status flags and their meanings

You can find out the status of a logical-log file using the methods explained in “Monitoring Logical-Log Files” on page 29-37.

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 they can be rolled forward if a data restore is needed.
- It makes logical-log file space available for new logical-log records.

You perform a logical-log file backup using either **ON-Archive** or **ontape**, depending on which of these utilities you are using to create and maintain your archives and backups.

Logical-log file backups can be initiated implicitly as part of continuous logging, or explicitly by the **OnLine** administrator or operator, either through ON-Archive or by executing **ontape**. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for more information.

When Are Logical-Log Files Freed?

If you back up a logical-log file, is that file free to receive new log records? Not necessarily. The following three criteria must be satisfied before **OnLine** 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 OnLine Attempt to Free a Log File?

OnLine attempts to free logical log files under the following conditions:

- When **OnLine** first writes to a new logical-log file, it attempts to free the previous log.
- Each time a logical-log file is backed up, **OnLine** attempts to free the backed-up log file.
- Each time **OnLine** commits or rolls back a transaction, it attempts to free the logical-log file in which the transaction began.

The attempt only succeeds if the criteria listed above (under “When Are Logical-Log Files Freed?”) are met.

What Happens If the Next Logical-Log File Is Not Free?

If **OnLine** attempts to switch to the next logical-log file and finds that the next log file in sequence is still in use, **OnLine** immediately suspends all processing. Even if other logical-log files are free, **OnLine** cannot skip a file in use and write to a free file out of sequence. Processing is stopped to protect the data within the log file.

The logical-log file might be in use for either of the following two reasons:

- The file is not backed up.

If the log file is not backed up, processing resumes when you perform the backup. If you are using **ontape** to back up logical-log files, you can back

up this log file as you would any other log file. If you are using ON-Archive to back up logical-log files, however, 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. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for more information.

- The file contains an open transaction.
The open transaction is the long transaction discussed under “Avoiding Long Transactions” on page 18-12. In this situation **OnLine** data has to be recovered from archive tapes in a full-system restore.

A situation whereby **OnLine** must suspend processing because the next log file contains the last checkpoint will never occur. **OnLine** always forces a checkpoint when it enters the last available log, if the previous checkpoint record is located in the log following the last available log. For example, if four logical-log files have the status shown in the following list, **OnLine** 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 **OnLine** needs to reuse that same logical-log file. In other words, it is a transaction that spans more than the total space allocated to the logical log.

Because **OnLine** 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.

How can you prevent long transactions from developing? You can 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 **OnLine** automatically slow down processing when a long transaction is developing.

These steps are explained in the subsequent sections.

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 **OnLine**, so you need to use your own judgment when estimating how quickly your logical log fills, and how to prevent long-transaction conditions.

- **Size of the Logical Log**
A smaller logical log fills faster than a larger logical log. If you need to make the logical log larger, you can add another logical-log file, explained in “Adding a Logical-Log File” on page 19-3.
- **Number of Logical-Log Records**
The more logical-log records written to the logical log, the faster it fills. If databases managed by your **OnLine** database server use transaction logging, transactions against those databases fill the logical log faster than transactions against databases without transaction logging.
- **Type of Log Buffering**
As explained in “Unbuffered Transaction Logging” on page 16-8, databases using unbuffered transaction logging fill the logical log faster than databases using 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 **OnLine** 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 and after images. Inserts store only the after image, and deletes store only the before image.
- **Frequency of Rollbacks**
The frequency of rollbacks affects the rate at which the logical log fills. More rollbacks fill the logical log faster. The rollbacks themselves require

logical-log file space, although the rollback records are small. In addition, rollbacks increase the activity in the logical log.

Factors That Prevent Closure of Transactions

Several factors influence when transactions close. You should 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, if a user running an interactive application leaves a terminal to go to lunch part of the way through a transaction.)

The more logical-log space is available, the longer a transaction can remain open without a long-transaction condition developing. However, 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 can delay the transaction

The amount of CPU activity can affect the ability of **OnLine** to complete the transaction. Repeated writes to the logical-log file increase the amount of CPU time **OnLine** 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

OnLine alters processing at two critical points to manage the long transaction condition. Both of the points are tunable by setting values in the ONCONFIG file.

The first critical point is called the *long-transaction high-water mark* and is described in “LTXHWM” on page 35-25. When the logical log reaches the long-transaction high-water mark, **OnLine** 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, **OnLine** 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 called the *exclusive-access, long-transaction high-water mark* and is described in “LTXEHWM” on page 35-25. When the logical log reaches the exclusive-access, long-transaction high-water mark, **OnLine** dramatically reduces log-record generation. Most threads are denied access to the logical log. Only threads currently rolling back transactions (including the long transaction) and threads currently writing COMMIT records are allowed access to the logical log. The intent is to preserve as much space as possible for rollback records 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, **OnLine** shuts down. If this 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.

It is imperative that the LTXHWM and LTXEHWM are set so **OnLine** does not come off-line due to a long transaction.

What Are the Logical-Log Administration Tasks Required for Blobspaces?

The following logical-log administration tasks are necessary if you have applications using blobspaces:

- Switching log files to activate blobspaces
- Switching log files to activate new blob space chunks
- Backing up log files to free deleted blob space pages

These tasks are explained in the following sections.

Switching Logical-Log Files to Activate Blobspaces

You must switch to the next logical-log file after you create a blob space if you intend to insert blobs in the blob space right away. **OnLine** requires that the statement that creates a blob space and the statements that insert blobs into that blob space appear in separate logical-log files. This requirement is independent of the logging status of the database.

See “Switching to the Next Logical-Log File” on page 19-12 for instructions on switching to the next log file.

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. **OnLine** 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.

See “Switching to the Next Logical-Log File” on page 19-12 for instructions on switching to the next log file.

Backing Up Logical-Log Files to Free BlobPages

When you delete data stored in blobSpace pages, those pages are not necessarily freed for reuse. The blobSpace pages are only free when *both* of the following actions have occurred:

- The blob has been deleted, either through an UPDATE to the blob column on the row, or by deleting the row.
- The logical log with the INSERT of the row containing the blob is backed up.

The reasons for this functionality are given in the following sections.

Why Do You Have to Back-Up Logical-Log Files to Free BlobPages?

Blobs that are stored in blobSpaces generally require much greater amounts of disk space than other data types. Because of this, if an application inserts a row containing a blob, **OnLine** does not write the actual blob data to the logical-log file. **OnLine** only writes the blobSpace overhead pages (which include the free-map page and the bit-map page) to the logical log.

The free-map page contains an entry for each blobPage in the blobSpace chunk. Each entry contains the following information:

- A flag indicating whether the page is used or free
- The unique id of the logical-log file that was current when the page was written to
- The associated tblSpace number of the data stored on the page

For more information on the free-map page, see “BlobSpace Page Types” on page 40-59.

When an application deletes a row containing a blob, **OnLine** marks each blobpage that is part of the deleted blob as FREE in the free-map overhead page. **OnLine** writes a log record to the logical log to record the changes to the free-map page. If **OnLine** has not yet backed up the logical-log file containing the transaction that originally inserted the blob, **OnLine** does not write over the deleted blobpages on subsequent blob inserts. Figure 18-3 illustrates this scenario.

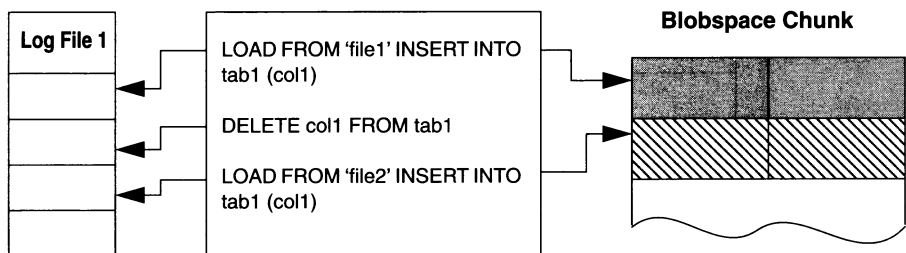


Figure 18-3 Schematic showing that **OnLine** does not write to deleted blobpages if logical-log file with original insert of blob is not backed up

The reason that **OnLine** does not write over the deleted blobpages in the scenario shown in Figure 18-3 is that **OnLine** might need to perform a fast recovery and roll forward the transaction containing the original insert. If the original data were overwritten, **OnLine** would have no way of reproducing the transaction.

When you back up the logical-log file containing the original insert, however, **OnLine** copies the actual blob data to tape. Because the blob data is saved to tape and, therefore, available for a recovery, **OnLine** is free to write over the deleted blobpages. (See Figure 18-4.)

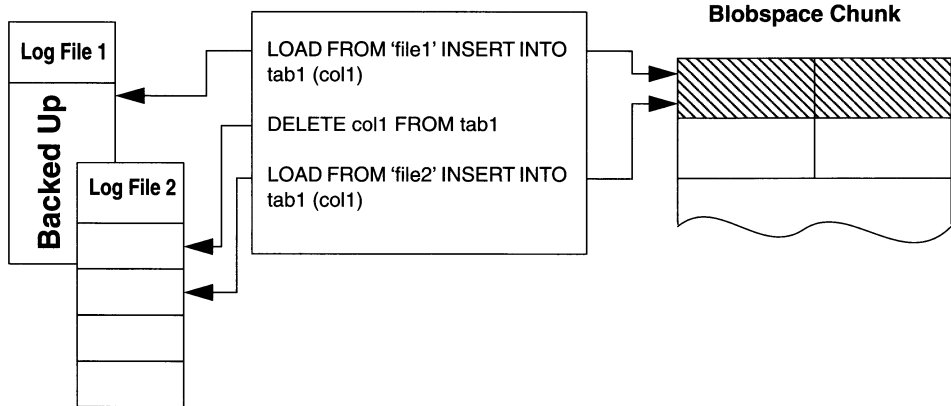


Figure 18-4 Schematic showing that OnLine can write to deleted blobpages if the logical-log file with the original insert of the blob is backed up

What Is the Logging Process?

This section describes the logging process in detail for both dbspace and blobspace logging. This information is not required for performing normal OnLine administration tasks.

Dbspace Logging

The logging process OnLine uses for operations involving data stored in dbspaces is as follows:

1. Read the data page from disk to shared-memory page buffer.
2. Copy the unchanged page to the physical-log buffer.
3. Write the new data into the page buffer; 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 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. **OnLine** identifies the page on which the row resides and attempts to locate the page in the **OnLine** shared-memory buffer pool. If the page is not already in shared memory, **OnLine** reads the page from disk. This process is explained in more detail in “How an OnLine Thread Accesses a Buffer Page” on page 14-36.

Copy the Page Buffer into the Physical-Log Buffer

Before **OnLine** modifies a dbspace data page, it stores a copy of the unchanged page in the physical-log page buffer. **OnLine** eventually flushes the physical-log page buffer containing this *before-image* to the physical log on disk. The before image of the page plays a critical role in fast recovery. Until **OnLine** 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 14-39.

How does **OnLine** know if a page is already in the physical log? If the timestamp on the page is more recent than the timestamp for the last checkpoint, the page has been changed since the checkpoint and therefore is already in the physical log.

Read Data into Buffer; Create Logical-Log Record

The thread performing the modifications receives data from the application. After **OnLine** 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 14-44.

Flush Physical Log Buffer to the Physical Log

OnLine must flush the physical log buffer before flushing 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 14-39.

Flush Page Buffer

At some point after flushing the physical log buffer, **OnLine** 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. **OnLine** does not flush the data buffer as the transaction is committed. See “How OnLine Flushes Data to Disk” on page 14-39 for more information.

Flush Logical-Log Buffer

OnLine flushes the logical-log buffer, writing the logical-log records to the current logical-log file on disk. See “Flushing the Logical-Log Buffer” on page 14-44 for more information.

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 ensures that all open transactions can be rolled back, if required.

Blobspace Logging

OnLine logs blobspace data but the data does not pass through either shared memory or the logical-log files on disk. Data stored in a blobspace is copied 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 blobs are deleted from blobpages), but not the actual blob data. Blobspace blob data is only recorded in the logical log 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 blob requires more than one blobpage, links and pointers are created 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 blob data is not included in the record (but the information about where the blob data is placed is included by way of the overhead pages).
3. When a logical-log backup begins, **OnLine** uses the logical-log id number stored in the blobspace free-map page to determine which blobpages to copy to tape. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for more details on logical-log backups.

Managing Logical-Log Files

Chapter Overview	3
Adding a Logical-Log File	3
Adding a Log File Using ON-Monitor	4
Adding a Log File Using <code>onparams</code>	4
Adding a Log File with a New Size	5
Dropping a Logical-Log File	5
Dropping a Logical-Log File Using ON-Monitor	6
Dropping a Logical-Log File Using <code>onparams</code>	6
Moving a Logical-Log File to Another Dbspace	6
An Example of Moving Logical-Log Files	7
Changing the Size of Logical-Log Files	7
Changing Logical-Log Configuration Parameters	8
Changing LOGSIZE or LOGFILES	8
Changing LOGSIZE or LOGFILES Using ON-Monitor	9
Changing LOGSIZE or LOGFILES Using an Editor	9
Changing LOGSMAX, LTXHWM, or LTXEHWM	9
Changing LOGSMAX, LTXHWM, or LTXEHWM Using ON-Monitor	10
Changing LOGSMAX, LTXHWM, or LTXEHWM by Editing the ONCONFIG File	10
Freeing a Logical-Log File	10
Freeing a Log File with Status A	10
Freeing a Log File with Status U	11
Freeing a Log File with Status U-B	11

Freeing a Log File with Status U-C or U-C-L	11
Freeing a Log File with Status U-B-L	12
Switching to the Next Logical-Log File	12

Chapter Overview

This chapter contains information on managing the **INFORMIX-OnLine Dynamic Server** logical-log files. The following tasks are covered:

- 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.

See Chapter 18, “What Is the Logical Log?,” for background information regarding the logical log.

See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for instructions on backing up logical-log files.

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 archive (quiescent or on-line), and **OnLine** must be in quiescent mode to add a logical-log file.

The newly added log file(s) do not become available until you create a level-0 archive of the root dbspace. This requirement ensures that the archive copy of the reserved pages contains information about the current number of logical-log files. You should use the archiving tool you usually use to create the level-0 archive.

You can use either ON-Monitor or **onparams** to add the log file. You can only add a new log file with a different size than LOGSIZE using **onparams**. 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 LTXEHWM” on page 19-9) and reinitialize shared memory for the change to take effect.

You must be logged in as either **informix** or **root** to make this change.

Adding a Log File Using ON-Monitor

Bring **OnLine** 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 archive of the root dbspace. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for information on creating a level-0 archive.

Adding a Log File Using *onparams*

To add a logical-log file with a size specified by LOGSIZE to the dbspace called **logspace**, execute the following command:

```
% 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 archive of the root dbspace. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for information on creating a level-0 archive.

See “Add a Logical-Log File” on page 37-38 for reference information on adding a logical-log file using **onparams**.

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 archive of the root dbspace. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for information on creating a level-0 archive.

See “Add a Logical-Log File” on page 37-38 for reference information on adding a logical-log file using **onparams**.

Dropping a Logical-Log File

You can drop a log to increase the amount of the disk space available within a dbspace.

OnLine requires a minimum of three logical-log files at all times. (Log files that are newly added and have status A do not count towards 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 archive of the root dbspace. This ensures that the archive copy of the reserved pages contains information about the current number of logical-log files. This prevents **OnLine** 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. See “Monitoring Logical-Log Files” on page 29-37 for information on obtaining a display of the logical-log files and logid numbers.

You must be logged in as either **informix** or **root**, and **OnLine** must be in quiescent mode, in order to make this change.

Dropping a Logical-Log File Using ON-Monitor

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 archive of the root dbspace after your configuration reflects the desired number of logical-log files. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for information on creating a level-0 archive.

Dropping a Logical-Log File Using *onparams*

Execute the following command to drop a logical-log file with id number 21:

```
% onparams -d -1 21
```

Create a level-0 archive of the root dbspace after your configuration reflects the number of logical-log files you want. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for information on creating a level-0 archive.

See “Drop a Logical-Log File” on page 37-38 for reference information on dropping a logical-log file using *onparams*.

Moving a Logical-Log File to Another Dbspace

You might want to move a logical-log file for performance reasons, explained in “Where Should Logical-Log Files Be Located?” on page 18-8. See “Monitoring Logical-Log Files” on page 29-37 to find out the location of logical-log files.

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 easy to do, it can be time-consuming because you must create two separate level-0 archives of the root dbspace as part of the procedure. An example of the procedure is presented in “An Example of Moving Logical-Log Files” on page 19-7.

An Example of Moving Logical-Log Files

OnLine must be in quiescent mode to make these changes.

The procedure listed here provides an example of how to move six logical-log files from the **root** dbspace to another dbspace: **dbspace_1**.

1. Free all log files except the current log file. See “Freeing a Logical-Log File” on page 19-10.
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 example, the value of LOGSMAX must be greater than or equal to nine. Change the value of LOGSMAX, if necessary. See “Changing LOGSMAX, LTXHWM, or LTXEHWM” on page 19-9.

3. Drop all but three of the logical-log files. See “Dropping a Logical-Log File” on page 19-5.

You cannot drop the current log file. If you only have 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 19-3.

In this example, add six new log files to **dbspace_1**.

5. Create a level-0 archive of the root dbspace to make the new log files available to **OnLine**. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for information on creating a level-0 archive.
6. Switch the logical-log files to start a new current log file. See “Switching to the Next Logical-Log File” on page 19-12.
7. Back up the former *current log file* to free it. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for information on backing up logical-log files.
8. Drop the three log files that remain in the root dbspace. See “Dropping a Logical-Log File” on page 19-5.

Changing the Size of Logical-Log Files

You can change the size of logical-log files in two ways:

- Add a new log file of a different size

This change has no effect on LOGSIZE. However, the log files you add are available after the next level-0 archive of the root dbspace (instead of after reinitializing disk space). See “Adding a Log File with a New Size” on page 19-5.

- Change the LOGSIZE configuration parameter

Changing LOGSIZE changes the default size for all subsequent logical-log files added but is time consuming because it requires that you reinitialize disk space to see the change. See “Changing LOGSIZE or LOGFILES” on page 19-8.

Changing Logical-Log Configuration Parameters

The following configuration parameters affect the logical-log file and how OnLine works with it:

- LOGSIZE (described on page 35-21)
- LOGFILES (described on page 35-20)
- LOGMAX (described on page 35-21)
- LTXHWM (described on page 35-25)
- LTXEHWM (described on page 35-25)

The procedure for changing each of these parameters is explained in the following sections.

Changing LOGSIZE or LOGFILES

You can change LOGSIZE or LOGFILES in two ways:

- Using ON-Monitor
- Using an 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 a relatively difficult process. If you want to increase the number of log files, it is easier to add log files one at a time as discussed under “Adding a Logical-Log File” on page 19-3. Similarly, if you want to change the size of the log files, it might be easier to add new log files of the desired size and then drop the old ones.

You can change LOGSIZE or LOGFILES from within ON-Monitor or by editing the ONCONFIG file. You must be logged in as **root** or **informix** to change these configuration parameters.

Changing LOGSIZE or LOGFILES Using ON-Monitor

You can change the value of LOGSIZE or LOGFILES using ON-Monitor.

Unload all **OnLine** data. See “onunload: Transfer Binary Data in Page Units” on page 37-77. You cannot rely on archive 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 file labelled `Number of Logical Logs`. Proceed with **OnLine** disk-space initialization.

After **OnLine** disk space is reinitialized, re-create all databases and tables. Then reload all **OnLine** data. See “onload: Create a Database or Table” on page 37-18.

Changing LOGSIZE or LOGFILES Using an Editor

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 **OnLine** data. See “onunload: Transfer Binary Data in Page Units” on page 37-77. You cannot rely on archive tapes to unload and restore the data, because a restore returns the parameters to their previous value.

Reinitialize disk space using **oninit**. After **OnLine** disk space is reinitialized, re-create all databases and tables. Then reload all **OnLine** data. See “onload: Create a Database or Table” on page 37-18.

Changing LOGSMAX, LTXHWM, or LTXEHWM

There are two ways to change the value of LOGSMAX, LTXHWM, or LTXEHWM:

- 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 Using ON-Monitor

You can change LOGSMAX, LTXHWM, or LTXEHWM while **OnLine** is in on-line mode using ON-Monitor.

Select the Parameters menu, Shared-Memory option to change one or more of the values. ON-Monitor displays the current values.

Enter the new value for LOGSMAX in the **Max # of Logical Logs** field. Or, enter the new value for LTXHWM in the **Long TX HWM** field, or the new value for LTXEHWM in **Long TX HWM Exclusive** field.

Reinitialize shared memory for the change or changes to take effect. See “Adding a Segment to the Virtual Portion of Shared Memory” on page 15-16.

Changing LOGSMAX, LTXHWM, or LTXEHWM by Editing the ONCONFIG File

You can change the value of LOGSMAX, LTXHWM, or LTXEHWM by using an editor to edit the ONCONFIG file.

Change the value of the parameter you wish to change.

Re-initialize shared memory for the change to take effect. See “Adding a Segment to the Virtual Portion of Shared Memory” on page 15-16.

Freeing a Logical-Log File

For a description of what constitutes a free logical-log file, see “What Are the Status Flags of Logical-Log Files?” on page 18-9.

You might want to free a logical-log file for the following reasons:

- So **OnLine** does not stop processing
- To free the space used by deleted blobpages

The procedures for freeing log files vary, depending upon the status of the log file. Each procedure is described in the following sections. See “Monitoring Logical-Log Files” on page 29-37 to find out the status of logical-log files.

Freeing a Log File with Status A

If a log file is newly added (status A), create a level-0 archive of the root dbspace to activate the log file and make it available for use. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for information on creating a level-0 archive.

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 archiving and backup tool you usually use. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for information on backing up logical-log files.

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” on page 19-11 or “Freeing a Log File with Status U-B-L” on page 19-12.

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 **OnLine** to quiescent mode. See “Immediately from On-Line to Quiescent” on page 8-5. 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 you 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 using the archiving and backup tool you usually use. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for information on backing up logical-log files.

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.

If after following these steps the log file now has status U-B or U-B-L, refer to “Freeing a Log File with Status U-B” or “Freeing a Log File with Status U-B-L.”

Freeing a Log File with Status U-B-L

If a log file is backed up to tape and all transactions within it are closed but the file is not free (status U-B-L), this logical-log file contains the most-recent checkpoint record.

To free log files with a status U-B-L, **OnLine** 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 switch log files to activate new blobspaces
- To switch log files to activate new blobspace chunks
- To switch log files to back up the current log

OnLine 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?

Chapter Overview 3

What Is Physical Logging? 3

 What Is the Purpose of the Physical Logging? 3

 Fast Recovery Uses Physically Logged Pages 4

 On-Line Archiving Uses Physically Logged
 Pages 4

 What OnLine Activity Is Physically Logged? 4
 Are Blobs Physically Logged? 4

What Is the Physical Log? 5

 How Big Should the Physical Log Be? 5

 Can the Physical Log Become Full? 6

 Where Is the Physical Log Located? 7

Details of Physical Logging 8

 Page Is Read into the Shared-Memory Buffer Pool 8

 A Copy of the Page Buffer Is Stored in the Physical-Log
 Buffer 8

 Change Is Reflected in the Data Buffer 9

 Physical-Log Buffer Is Flushed to the Physical Log 9

 Page Buffer Is Flushed 9

 When Checkpoint Occurs, Physical-Log Buffer Is
 Flushed and Physical Log Is Emptied 9

 How the Physical Log Is Emptied 9

Chapter Overview

This chapter defines the terms and explains the concepts you need to know to effectively perform the tasks described in Chapter 21, “Managing the Physical Log.”

This 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-OnLine Dynamic Server** is going to change before the changed pages are actually recorded. Before **OnLine** 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. **OnLine** maintains the before-image page in the physical-log buffer in shared memory for those pages until they are flushed to disk by one of four occurrences. (See “OnLine Checkpoints” on page 14-47.) Once a checkpoint occurs, **OnLine** empties the physical log (except in the special circumstances explained in “Can the Physical Log Become Full?” on page 20-6).

What Is the Purpose of the 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 archiving procedure needs them to provide an accurate snapshot of **OnLine** data. These snapshots are potentially used in two activities: fast recovery and **OnLine** archiving.

Fast Recovery Uses Physically Logged Pages

After a failure, **OnLine** 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, **OnLine** can return all data to physical and logical consistency, up to the point of the most recently completed transaction. This procedure is explained in more detail in “What Is Fast Recovery?” on page 22-3.

On-Line Archiving Uses Physically Logged Pages

When you perform an on-line archive, **OnLine** checks disk pages to see which should be archived. As part of this process, **OnLine** periodically reads the pages in the physical log. If any of the pages **OnLine** finds in the physical log meet the archiving criterion, they are copied to the archive tape. For details, see the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*.

What OnLine Activity Is Physically Logged?

All dbspace page modifications except the following ones are physically logged:

- Pages that do not have a valid **OnLine** address. This situation usually occurs when the page was used by some other **OnLine** database server or a table that was dropped.
- Pages that **OnLine** 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. You must remember, however, that **OnLine** stores the before-images in the physical log for a short period of time, that is, until the next checkpoint. You can control the amount of data that **OnLine** logs by tuning the checkpoint interval configuration parameter, CKPTINTVL.

Are Blobs Physically Logged?

The **OnLine** pages in the physical log can be any **OnLine** page except a blob-space blobpage. 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 Blobs Logged?” on page 16-7.

What Is the Physical Log?

The physical log is a set of contiguous disk pages where **OnLine** stores before-images.

How Big Should the Physical Log Be?

When considering 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 **OnLine** needs in a worst case scenario. This scenario takes place when a checkpoint occurs because of the log becoming 75 percent full. If all of the update threads are in a critical section (see “Critical Sections” on page 14-46) and perform physical logging of the maximum number of pages in their critical section, **OnLine** must fit this logging into the final 25 percent of the physical log so as to prevent a physical-log overflow.

The maximum number of pages (`max_log_pages_per_critical_section`) that **OnLine** 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.

$$\text{USERTHREADS} * \text{maximum log pages per critical section}$$

The exception to this rule occurs if you are using `tblspace` blobs in a database without logging. Here, you should substitute the size of the most frequently occurring blob in the `dbspace` for the maximum log pages per critical section.

In addition you need to think about the following issues:

- How much updating of data does **OnLine** perform?

Operations that do not perform updates do not generate before-images, so if the applications using 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.

You should also keep in mind that **OnLine** only writes the before-image of the first update made to a page. Thus, if your application is repeatedly updating 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?

Since 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 check point interval, CKPTINTVL, so that checkpoints occur more frequently. Be aware, however, that decreasing the checkpoint interval beyond a certain point has an impact on performance.

If you plan to increase the checkpoint interval, or 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 35-33.)

Can the Physical Log Become Full?

Since a checkpoint is initiated that logically empties the physical log when it becomes 75 percent full, it is unlikely that it would become 100 percent full. You can further assure that the physical log does not become full during a checkpoint by taking the following actions:

- Configuring **OnLine** according to the sizing guidelines for the physical log and the logical log files
- Fine tuning the size of the physical log as a result of monitoring it during production activity

However, it is still possible that the physical log could 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 14-46.) However, threads *currently in* critical sections can continue processing. It is possible for the physical log to become full if many threads in critical sections are processing work *and* if the space remaining 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 **OnLine** processes tblspace blobs stored in a database created with transaction logging, each portion of the blob that **OnLine** 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, **OnLine** 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 occurs, **OnLine** sends the message

Physical log file overflow

to the message log and then initiates a shutdown. See this message in your message log for suggested corrective action.

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 18-12.) 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, it is conceivable that the writes performed as threads complete their processing could 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 the LTXEHW was reached.

Where Is the Physical Log Located?

When **OnLine** disk space is initialized, the logical log files and the physical log are placed 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 is not shared by active tables or the logical log files.

The dbspace in which the physical log is located is specified in the ONCONFIG parameter PHYSDBS. (See “PHYSDBS” on page 35-33). You should change this parameter only if you decide to move the physical-log file from the root dbspace. (See “Changing the Physical-Log Location and Size” on page 21-3.)

Because of the critical nature of the physical log, Informix recommends that you mirror the dbspace that contains the physical log.

Details of Physical Logging

This section describes the details of physical logging. It is provided to satisfy your curiosity; you do not need to understand the information here to manage your physical log.

OnLine 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.

Each step is explained in detail in the paragraphs that follow.

Page Is Read into the Shared-Memory Buffer Pool

When a session requests a row, **OnLine** identifies the page on which the row resides and attempts to locate the page in the **OnLine** shared-memory buffer pool. If the page is not already in shared memory, it is read into the resident portion of **OnLine** 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. **OnLine** reflects these changes in the shared-memory data buffer.

Data from the application is passed to **OnLine**. 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

OnLine will most likely 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 **OnLine** Achieves Data Consistency" on page 14-46.

When Checkpoint Occurs, Physical-Log Buffer Is Flushed and Physical Log Is Emptied

A checkpoint can occur at any point in the physical logging process. After a checkpoint occurs, **OnLine** 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. **OnLine** empties the physical log logically, allowing current entries to be overwritten.

How the Physical Log Is Emptied

OnLine 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

Chapter Overview 3

Changing the Physical-Log Location and Size 3

Why Change Physical-Log Location and Size? 3

Before You Make the Changes 4

Using ON-Monitor to Changing Physical-Log Location
or Size 4

Using an Editor to change Physical-Log Location and
Size 5

Using *onparams* to Change Physical-Log Location or
Size 5

Chapter Overview

This chapter describes how to change the configuration parameters associated with the physical log. For background information about the physical log, see Chapter 20, “What Is Physical Logging?”

Changing the Physical-Log Location and Size

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

You must be logged in as user **informix** or **root** when you make the changes. Each of these methods is described in the following sections.

For any of the three methods, you can activate the changes to the size or location of the physical log as soon as you make them by reinitializing shared memory. If you use **onparams**, you can reinitialize shared memory in the same step.

You should create a level-0 archive immediately after you reinitialize shared memory. This archive is critical for **INFORMIX-OnLine Dynamic Server** recovery. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*.

Why Change Physical-Log Location and Size?

You can move the physical-log file to try to improve performance. When **OnLine** disk space is initialized, the disk pages allocated for the logical log and the physical log are always located in the root dbspace. You might be able to improve performance by moving the physical log or the logical-log files or both to other dbspaces.

See “Where Is the Physical Log Located?” on page 20-7 for advice on where to place the physical log, and “How Big Should the Physical Log Be?” on page 20-5 for advice on sizing the physical log.

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 **OnLine**.

You can check if there is adequate contiguous space with the **-pe** option of the **oncheck** utility. See “-pe Option” on page 37-13 for more information.

You must be logged in as user **root** or **informix** to change the physical-log location or size using either ON-Monitor or **onparams**.

Using ON-Monitor to Changing 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 and, second, if you want to shut **OnLine** down. This last message refers to reinitializing shared memory. If you respond **Y**, ON-Monitor reinitializes shared memory and any changes are implemented immediately. 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 archive immediately to ensure that all recovery mechanisms are available.

Using an 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 **OnLine** is in on-line mode:

- **PHYSFILE** page 35-33
- **PHYSDBS** page 35-33

The changes do not take effect until you reinitialize shared memory.

After you reinitialize shared memory, create a level-0 archive 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 37-37.

To change the size and location of the physical log, execute the following command after you bring **OnLine** to quiescent mode:

```
% onparams -p -s size -d dbspace -y
```

where *size* is the new size of the physical log in kilobytes, and *dbspace* specifies the new dbspace where the physical log is to reside.

The following example changes the size and location of the physical log. The new physical-log size is 400 kilobytes, and the log will reside in the **dbspace6** dbspace. The command also reinitializes shared memory with the **-y** option so that the change takes effect immediately, as follows:

```
% onparams -p -s 400 -d dbspace6 -y
```

After you reinitialize shared memory, create a level-0 archive to ensure that all recovery mechanisms are available. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*.

What Is Fast Recovery?

Chapter Overview 3

What Is Fast Recovery? 3

When Is Fast Recovery Needed? 3

When Does OnLine Initiate Fast Recovery? 4

Fast Recovery and Buffered Logging 4

Fast Recovery and No Logging 4

Details of Fast Recovery 5

Return to the Last-Checkpoint State 5

Find the Checkpoint Record in the Logical Log 6

Roll Forward Logical Log Records 7

Roll Back Incomplete Transactions 8

Chapter Overview

You do not need to take any administrative actions with respect to fast recovery; it is an automatic feature. You might want to read this chapter, however, if you are interested in what fast recovery is and how it works.

This chapter covers the following topics:

- A definition of fast recovery, including the types of failures fast recovery addresses and how **INFORMIX-OnLine Dynamic Server** detects these failures
- The details of how fast recovery works

What Is Fast Recovery?

Fast recovery is an automatic, fault-tolerance feature that **OnLine** executes every time **OnLine** moves from off-line to on-line mode or from quiescent to on-line mode.

The fast-recovery process then checks to see if the last time **OnLine** went off-line, it did so in uncontrolled conditions. If so, fast recovery returns **OnLine** to a state of physical and logical consistency as described in “Details of Fast Recovery” on page 22-5.

If the fast-recovery process finds that **OnLine** came off-line in a controlled manner, the fast-recovery process terminates and **OnLine** moves to on-line mode.

When Is Fast Recovery Needed?

Fast recovery restores **OnLine** to physical and logical consistency after any failure that results in the loss of the contents of memory for **OnLine**. Such failures are usually termed system failures. System failures do not cause any physical damage to the database but instead affect transactions that are in progress at the time of the failure.

Fast recovery addresses system failures like the following example: **OnLine** is processing tasks for more than 40 users. Dozens of transactions are on-going. Without warning, the operating system fails.

How does **OnLine** bring itself to a consistent state again? What happens to ongoing transactions? The answer to both questions is fast recovery.

When Does OnLine Initiate Fast Recovery?

OnLine checks if fast recovery is needed every time the administrator brings **OnLine** to quiescent mode or on-line mode from off-line mode.

As part of shared-memory initialization, **OnLine** checks the contents of the physical log. The physical log is empty when **OnLine** shuts down under control. The move from on-line mode to quiescent mode includes a checkpoint, which flushes the physical log. Therefore, if **OnLine** finds pages in the physical log, it is clear **OnLine** went off-line under uncontrolled conditions, and fast recovery begins.

Fast Recovery and Buffered Logging

If a database uses buffered logging (as described in “Buffered Transaction Logging” on page 16-9), some logical log records associated with committed transactions might not be written to the logical log at the time of the failure. If this occurs, fast recovery is unable to restore those transactions. Fast recovery can only restore transactions with an associated COMMIT record stored in the logical log on disk. (This is why 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

The result of fast recovery is to return **OnLine** to a consistent state as part of shared-memory initialization. The consistent state means that all committed transactions are restored and all uncommitted transactions are rolled back.

Fast recovery is accomplished in the following two stages:

- The physical log is used to return **OnLine** to the most-recent point of known *physical consistency*, the most-recent checkpoint.
- The logical log files are used to return **OnLine** to *logical consistency*, by rolling forward all committed transactions that have occurred since the 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. Return all disk pages to their condition at the time of the most-recent checkpoint using the data in the physical log.
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.

Return to the Last-Checkpoint State

The first step, returning all disk pages to their condition at the time of the most-recent checkpoint is accomplished by writing 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. By writing each before-image page in the physical log to shared memory and then back to disk, changes to **OnLine** data since the time of the most-recent checkpoint are undone. Figure 22-1 illustrates this step.

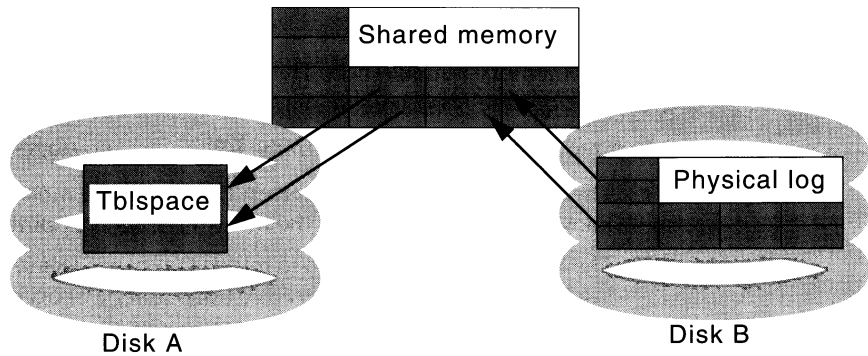


Figure 22-1 OnLine writes all before-images that remain in the physical log back to disk, returning the data to its state as of the most-recent checkpoint.

Find the Checkpoint Record in the Logical Log

The second step is to locate 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.

All 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 40-8.)

Once this information is read, it also identifies the location of all logical log records written after the most-recent checkpoint. Figure 22-2 illustrates this step.

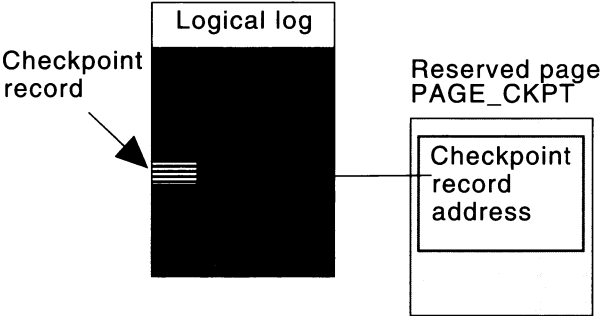


Figure 22-2 OnLine locates the most-recent checkpoint record in the logical log.

Roll Forward Logical Log Records

The third step in fast recovery is to roll forward the logical log records that were written after the most-recent checkpoint record. This action reproduces all changes to the databases since the time of the last checkpoint, up to the point where the uncontrolled shutdown occurred. Figure 22-3 illustrates this step.

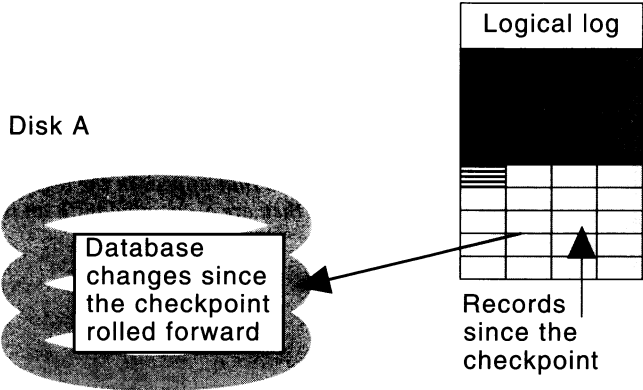


Figure 22-3 OnLine rolls forward the logical log records written since the most-recent checkpoint, reproducing the changes to the database since the checkpoint.

Roll Back Incomplete Transactions

The final step in fast recovery is to roll 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.) This rollback procedure ensures that all databases are left in a consistent state.

Because it is possible that one or more transactions 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 **OnLine** because a log file is not freed until all transactions contained within it are closed. Figure 22-4 illustrates the roll-back procedure. When fast recovery is complete, **OnLine** goes to quiescent or on-line mode.

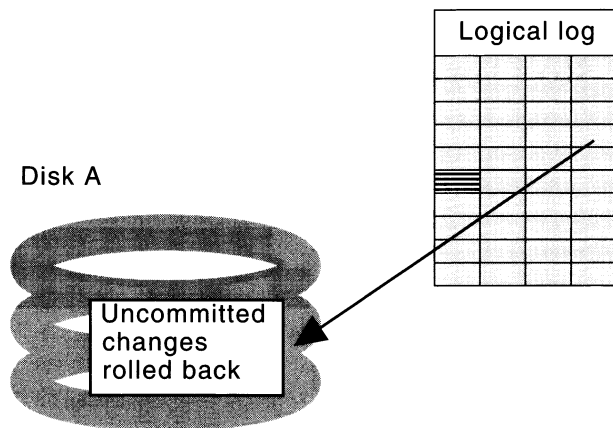


Figure 22-4 *OnLine rolls back all incomplete transactions, ensuring that all databases are left in a consistent state. Records that appear in the logical log earlier than the checkpoint might be rolled back if they are associated with an incomplete transaction.*



Fault Tolerance





What Is Mirroring?

Chapter Overview 3

What Is Mirroring? 3

What Are the Benefits of Mirroring? 4

What Are the Costs of Mirroring? 4

What Happens If You Do Not Mirror? 5

What Should You Mirror? 5

What Mirroring Alternatives Exist? 5

The Mirroring Process 6

What Happens When You Create a Mirror Chunk? 6

What Are Mirror Status Flags? 7

What Is Recovery? 7

What Happens During Processing? 8

Disk Writes to Mirrored Chunks 8

Disk Reads from Mirrored Chunks 8

Detecting Media Failures 9

Recovering a Chunk 9

What Happens If You Stop Mirroring? 10

What Is the Structure of a Mirror Chunk? 10

Chapter Overview

The first part of this chapter answers the following basic questions about the **INFORMIX-OnLine Dynamic 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 mirror 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 mirror chunk?

For instructions on how to perform mirroring tasks, refer to Chapter 24, "Using Mirroring."

What Is Mirroring?

Mirroring is a strategy that pairs a *primary* chunk of one defined dbspace or blobspace with an equal-sized *mirror* chunk. Every write to the primary chunk is automatically accompanied by an identical write to the mirror chunk. This concept is illustrated in Figure 23-1. If a failure occurs on the primary chunk, mirroring enables you to read from and write to the mirror chunk until you can recover the primary chunk, all without interrupting user access to data.

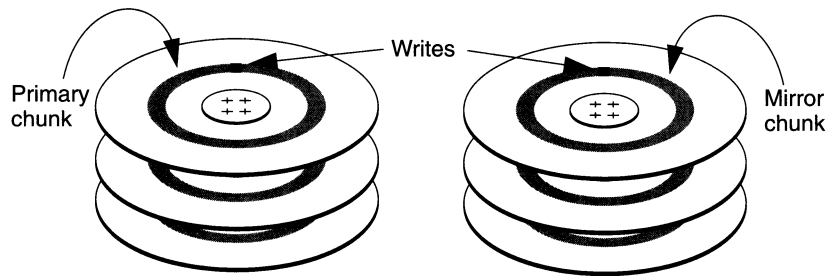


Figure 23-1 Data is written to both the primary chunk and the mirror chunk.

Mirroring on disks managed over a network is not supported. The same **OnLine** database server must manage all the chunks of a mirrored set.

What Are the Benefits of Mirroring?

Mirroring provides the **OnLine** administrator with a means of recovering data, in the event of a media failure, without having to take **OnLine** 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 **OnLine** 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 mirror chunks. The use of multiple virtual processors for disk writes reduces this performance cost. The use of *split reads*, whereby **OnLine** reads data from either the primary chunk or the mirror chunk depending on the location of the data within the chunk, actually causes performance to improve for read-only data. See “What Happens During Processing?” on page 23-8 for more information on how **OnLine** performs reads and writes for mirrored chunks.

What Happens If You Do Not Mirror?

If you do not mirror your dbspaces, the frequency with which you have to restore from an archive in the event of a media failure increases.

When a mirrored chunk suffers a media failure, **OnLine** 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, **OnLine** cannot access the data stored on that chunk. If the chunk contains logical-log files, the physical log, or the root dbspace, **OnLine** goes off-line immediately. If the chunk does not contain logical-log files, the physical log, or the root dbspace, **OnLine** 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 an archive.

What Should You Mirror?

Ideally you would be able to mirror all of your data. If disk space is an issue, however, you might not be able to do this. In this case, you should 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, because if these chunks go down **OnLine** goes off-line immediately.

If you have chunks that hold data that is critical to your business, you should also give these chunks high priority for mirroring.

You should also give priority for mirroring to other chunks that store data that is frequently used. This 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 an **OnLine** feature. Alternative mirroring solutions, provided by your operating system or hardware, might be available.

If you are considering a mirroring feature provided by your operating system instead of **OnLine** mirroring, compare the implementation of both features before you decide which to use. Bear in mind that the slowest step in the mirroring process is the actual writing of data to disk. The **OnLine** strategy of performing writes to mirrored chunks in parallel (see “Disk Writes to Mirrored Chunks” on page 23-8) helps to reduce the time required for this step. In addition, **OnLine** mirroring uses split reads to improve read performance.

(See “Disk Reads from Mirrored Chunks” on page 23-8.) Operating system mirroring features which do not use parallel mirror writes and split reads might give inferior performance.

Nothing prevents you from running **OnLine** mirroring and operating system mirroring at the same time. They run independently of each other. In some cases, you might decide to use both **OnLine** mirroring and the mirroring feature provided by your operating system. For example, you might have both **OnLine** data and non-**OnLine** data on a single disk drive. You could use the operating system mirroring to mirror the non-**OnLine** data and **OnLine** mirroring to mirror the **OnLine** 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 **OnLine** mirroring does to store the same amount of data in a manner resilient to media failure. The disadvantage is that it is slower than **OnLine** 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 mirror chunks, starting mirroring, changing the status of mirror chunks, and so on, refer to Chapter 24, “Using Mirroring.”

What Happens When You Create a Mirror Chunk?

When you specify a mirror chunk, **OnLine** copies all the data from the primary chunk to the mirror chunk. This copy process is known as *recovery*. Mirroring begins as soon as recovery is complete.

The recovery procedure that marks the beginning of mirroring is delayed if you start to mirror chunks within a dbspace that contains a logical-log file. Mirroring for dbspaces that contain a logical-log file does not begin until you create a level-0 archive of the root dbspace. The delay ensures that **OnLine** can use the mirrored logical-log files if the primary chunk containing these logical-log files becomes unavailable during an archive restore. The level-0 archive operation copies the updated **OnLine** configuration information,

including information about the new mirror chunk, from the root dbspace reserved pages to the archive. If you do a data restore, the updated configuration information at the beginning of the archive directs **OnLine** to look for the mirrored copies of the logical-log files if the primary chunk becomes unavailable. If this new archive information does not exist, **OnLine** is unable to take advantage of the mirrored log files.

For similar reasons, you cannot mirror a dbspace that contains a logical-log file while an archive is being created. The new information that must appear in the first block of the archive tape cannot be copied there once the archive has begun.

For more information on creating mirror chunks, refer to Chapter 24, "Using Mirroring."

What Are Mirror Status Flags?

Dbspaces and blobspaces have status flags that indicate whether the dbspace or blobspace is mirrored, unmirrored, or mirrored but requiring a level-0 archive of the root dbspace before mirroring starts.

Chunks have status flags that indicate the following information:

- Whether the chunk is a primary or mirror chunk
- Whether the chunk is currently on-line, down, a new mirror chunk requiring a level-0 archive of the root dbspace, or being recovered.

For descriptions of these chunk status flags, refer to "-d Option" on page 37-52. For information on how to display these status flags, refer to "Monitoring Chunk Status" on page 29-43.

What Is Recovery?

When **OnLine** recovers a mirrored chunk, it performs the same recovery procedure 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, **OnLine** 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 mirror chunk.

Note that 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, **OnLine** updates the corresponding page on the recovery chunk before continuing with the recovery process.

For information on how to recover a down chunk, refer to “Recovering a Mirrored Chunk” on page 24-9.

What Happens During Processing?

This section discusses some of the details of disk I/O for mirrored chunks and how **OnLine** handles media failure for these chunks.

Disk Writes to Mirrored Chunks

During **OnLine** processing, **OnLine** performs mirroring by executing two writes for each modification: one to the primary chunk and one to the mirror chunk. Virtual processors of the AIO class perform the actual disk I/O. For more information, refer to “Asynchronous I/O” on page 12-22.

The requesting thread submits the two write requests (one for the primary chunk and one for the mirror 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

OnLine makes use of mirroring to improve read performance because two versions of the data reside on separate disks. A data page is read from either the primary chunk or the mirror chunk, depending on which half of the chunk includes the address of the data page. This 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. This is illustrated in Figure 23-2.

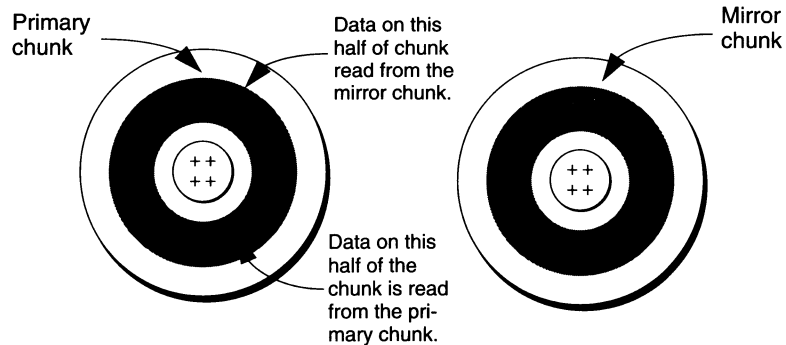


Figure 23-2 Schematic illustrating how split reads reduce the maximum distance over which the disk head must travel

Detecting Media Failures

OnLine checks the return code when it first opens a chunk and after any read or write. Whenever **OnLine** detects that a primary (or mirror) chunk device has failed, it sets the chunk status flag to down (D). Refer to “What Are Mirror Status Flags?” on page 23-7 for information on chunk status flags.

If **OnLine** 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 mirror chunks and writes are made to both chunks.

Recovering a Chunk

OnLine 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 performing the read does not have to wait until the thread performing the write has finished before reading more data.

What Happens If You Stop Mirroring?

When you end mirroring, **OnLine** immediately frees the mirror chunks and makes the space available for reallocation. The action of ending mirroring takes only a few seconds.

You should create a level-0 archive of the root dbspace after you end mirroring to ensure that the reserved pages with the updated mirror chunk information are copied to the archive. This prevents the restore procedure from assuming that mirrored data is still available.

What Is the Structure of a Mirror Chunk?

The mirror chunk contains the same control structures as the primary chunk. Mirrors of blob space chunks contain blob space control pages; mirrors of db space chunks contain db space control pages. Refer to “Structure of a Mirror Chunk” on page 40-15 for information on these structures.

A display of disk-space use, given by one of the methods discussed under “Monitor Chunks” on page 29-46, always indicates that the mirror chunk is full, even if the primary chunk has free space. The *full* mirror chunk indicates that none of the space in the chunk is available for use other than as a mirror of the primary chunk. The status remains full for as long as both primary chunk and mirror chunk are on-line.

If the primary chunk goes down and the mirror chunk becomes the primary chunk, disk-space allocation reports then accurately describe the fullness of the new primary chunk.

Using Mirroring

- Chapter Overview 3
- Steps Required for Mirroring Data 3
- Enabling Mirroring 4
 - Enabling Mirroring Using ON-Monitor 4
 - Enabling Mirroring by Editing the ONCONFIG File 5
- Allocating Disk Space for Mirrored Data 5
- Starting Mirroring 5
 - Mirroring the Root Dbspace During Initialization 6
 - Setting MIRRORPATH and MIRROROFFSET Using ON-Monitor 6
 - Setting MIRRORPATH and MIRROROFFSET Using a Text Editor 6
 - Starting Mirroring for Unmirrored Dbspaces 6
 - Starting Mirroring for Unmirrored Dbspaces Using ON-Monitor 7
 - Starting Mirroring for Unmirrored Dbspaces Using the **onspaces** Utility 7
 - Starting Mirroring for New Dbspaces 7
 - Starting Mirroring for New Dbspaces Using ON-Monitor 7
 - Starting Mirroring for New Dbspaces Using the **onspaces** Utility 7
- Adding Mirror Chunks 8
 - Adding Mirror Chunks Using ON-Monitor 8
 - Adding Mirror Chunks Using the **onspaces** Utility 8

Changing the Mirror Status	8
Taking Down a Mirror Chunk	8
Taking Down a Mirror Chunk Using ON-Monitor	9
Taking Down a Mirror Chunk Using the onspaces Utility	9
Recovering a Mirrored Chunk	9
Recovering a Mirrored Chunk Using ON-Monitor	9
Recovering a Mirrored Chunk Using the onspaces Utility	9
Relinking a Chunk to a Device After a Disk Failure	10
Ending Mirroring	10
Ending Mirroring Using ON-Monitor	10
Ending Mirroring Using onspaces	11

Chapter Overview

This chapter describes the various mirroring tasks that are necessary to make use of the **INFORMIX-OnLine Dynamic Server** mirroring feature. First an overview is given of the steps required for mirroring data. Then the following tasks are described:

- Enabling mirroring
- Allocating disk space for mirror chunks
- Starting mirroring (creating mirror 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 **OnLine** off-line and enable mirroring. (See “Enabling Mirroring” on page 24-4.)
2. Reinitialize shared memory.
3. Allocate disk space for the mirror chunks. You can allocate this disk space at any time, as long as the disk space is available when you specify mirror chunks in the next step. (See “Allocating Disk Space for Mirrored Data” on page 24-5.)
4. Choose the dbspace that you want to mirror and create mirror chunks by specifying a mirror-chunk pathname and offset for each primary chunk in that dbspace. The mirroring process starts after you perform this step. Repeat this step for all the dbspaces that you want to mirror. (See “Starting Mirroring” on page 24-5.)

Enabling Mirroring

When you enable mirroring, you invoke the **OnLine** functionality required for mirroring tasks. However, when you enable mirroring, you do not initiate the mirroring process. Mirroring does not actually start until you create mirror chunks for a dbspace or blobspace. (See “Starting Mirroring” on page 24-5.)

To enable mirroring for **OnLine**, you must set the MIRROR parameter in ONCONFIG to one. The default value of MIRROR is zero, indicating mirroring is disabled.

Enable mirroring when you initialize **OnLine** 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 **OnLine** is in on-line mode, but it does not take effect until you reinitialize shared memory (take **OnLine** 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, because you might accidentally reinitialize your disk if you are not careful when enabling mirroring with ON-Monitor.

Enabling Mirroring Using ON-Monitor

Select the Parameters menu, Initialize option to enable mirroring. In the field labelled `MIRROR`, enter a `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 **OnLine** disk space and destroy all existing data). Respond `N` (no) to this prompt.



Warning: *If you respond `Y` (yes) at this prompt, you will lose all of your existing data.*

Reinitialize shared memory (take **OnLine** 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 **OnLine** off-line and then to quiescent mode) for the change to take effect.

Allocating Disk Space for Mirrored Data

Before you can create a mirror chunk, you must allocate disk space for this purpose. You can allocate either raw disk space or cooked file space for mirror chunks. For a discussion of allocating disk space, refer to “Allocating Disk Space” on page 11-3.

Always allocate disk space for a mirror chunk on a different disk than the corresponding primary chunk with, ideally, a different controller. This setup allows you to access the mirror chunk if the disk on which the primary chunk is located goes down, or vice versa.

Use the UNIX link (**ln**) command to link the actual files or raw devices of the mirror chunks to mirror pathnames. In the event of disk failure, you can link a new file or raw device to the pathname, eliminating the need to physically replace the disk that failed before the chunk is brought back on-line. (See “Relinking a Chunk to a Device After a Disk Failure” on page 24-10.)

Starting Mirroring

Mirroring starts when you create a mirror 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 mirror chunk. You can use either ON-Monitor or the **onspaces** utility to create mirror chunks.

When you create a mirror chunk, **OnLine** performs the *recovery* process, copying data from the primary chunk to the mirror chunk. When this process is complete, **OnLine** begins mirroring data. If the primary chunk contains logical-log files, **OnLine** does not perform the recovery process immediately after you create the mirror chunk but instead waits until you perform a level-0 archive. See “What Happens When You Create a Mirror Chunk?” on page 23-6 for an explanation of this behavior.

You must always start mirroring for an entire dbspace or blobspace. **OnLine** 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 mirror 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 mirror chunks for the existing chunks in the dbspace or blobspace. You can perform all three operations using ON-Monitor, and you can perform the last two using **onspaces** as well.

Mirroring the Root Dbspace During Initialization

If you enable mirroring when you initialize **OnLine**, you can also specify a mirror pathname and offset for the root chunk. **OnLine** creates the mirror chunk when it is initialized. However, since the root chunk contains logical log files, mirroring does not actually start until you perform a level 0 archive. (See “What Happens When You Create a Mirror Chunk?” on page 23-6.)

You specify the root mirror pathname and offset by setting 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 **OnLine** is initialized. (See “Starting Mirroring for Unmirrored Dbspaces” on page 24-6.)

Setting **MIRRORPATH** and **MIRROROFFSET** Using **ON-Monitor**

If you are using **ON-Monitor** to initialize **OnLine**, you can set the **MIRRORPATH** and **MIRROROFFSET** parameters in the **DISK PARAMETERS** screen of the Parameters menu, Initialize option.

Setting **MIRRORPATH** and **MIRROROFFSET** Using a Text Editor

If you are using **oninit** to initialize **OnLine**, you must use a text editor to set the values of **MIRRORPATH** and **MIRROROFFSET** in **ONCONFIG** before you bring up **OnLine**.

Starting Mirroring for Unmirrored Dbspaces

You can start mirroring for any dbspace or blobspace using either **ON-Monitor** or the **onspaces** utility.

Starting Mirroring for Unmirrored Dbspaces Using ON-Monitor

Use the Mirror option of the Dbspaces menu to start mirroring a dbspace. The first screen displays a list of dbspaces. Select the dbspace you want to mirror by moving the cursor down the list to the correct dbspace and typing 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. **OnLine** recovers the new mirror chunks unless they contain logical log files, in which case recovery is postponed until after you do a level-0 archive.

Starting Mirroring for Unmirrored Dbspaces Using the *onspaces* Utility

You can also use the **onspaces** utility to start mirroring a dbspace or blob-space. 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
```

See “*onspaces: Modify Blobspaces or Dbspaces*” on page 37-40 for a full description of the **onspaces** syntax.

Starting Mirroring for New Dbspaces

You can also start mirroring when you create a new dbspace or blob-space. You can use either ON-Monitor or the **onspaces** utility to do this.

Starting Mirroring for New Dbspaces Using 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 mirror chunk for the new dbspace.

Starting Mirroring for New Dbspaces Using the *onspaces* Utility

You can use the **onspaces** utility to create a mirrored dbspace. For example, the following command creates the dbspace **db_acct** with an initial chunk **/dev/chunk1** and a mirror chunk **/dev/mirror_chk1**:

```
% onspaces -c -d db_acct -p /dev/chunk1 -o 0 -s 2500 -m /dev/mirror_chk1 0
```

See “*onspaces: Modify Blobspaces or Dbspaces*” on page 37-40 for a full description of the **onspaces** syntax.

Adding Mirror Chunks

If you add a chunk to a dbspace that is mirrored, you must also add a corresponding mirror chunk.

Adding Mirror Chunks Using 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 Mirror Chunks Using the *onspaces* Utility

You can also use the **onspaces** utility to add a primary chunk and its mirror chunk to a dbspace. The following example adds a chunk, **chunk2**, to the **db_acct** dbspace. Since the dbspace is mirrored, a mirror chunk, **mirror_chk2**, is also added.

```
% onspaces -a db_acct -p /dev/chunk2 -o 5000 -s 2500 -m /dev/mirror_chk2 5000
```

See “onspaces: Modify Blobspaces or Dbspaces” on page 37-40 for a full description of the **onspaces** syntax.

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 mirror 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 29-43.

Taking Down a Mirror Chunk

When a mirror chunk is *down*, **OnLine** cannot write to it or read from it. You might take down a mirror chunk to relinking the chunk to a different device (see “Relinking a Chunk to a Device After a Disk Failure” on page 24-10.)

Taking down a chunk is not the same as ending mirroring. You end mirroring for a complete dbospace, which causes **OnLine** to drop all the mirror chunks for that dbospace.

Taking Down a Mirror Chunk Using ON-Monitor

To use ON-Monitor to take down a mirror chunk, choose the Status option from the Dbspaces menu. With the cursor on the dbospace that contains the chunk that you want to take down, press F3 or CTRL-B. **OnLine** displays a screen listing all of the chunks in the dbospace. Move the cursor to the chunk you wish to take down and type F3 or CTRL-B to change the status (take it down).

Taking Down a Mirror Chunk Using 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 dbospace **db_acct**:

```
% onspaces -s db_acct -p /dev/mirror_chk1 -o 0 -D
```

See “*onspaces: Modify Blobspaces or Dbspaces*” on page 37-40 for a full description of the **onspaces** syntax.

Recovering a Mirrored Chunk

You recover a down chunk to begin mirroring the data on the chunk that is on-line.

Recovering a Mirrored Chunk Using ON-Monitor

To use ON-Monitor to recover a down chunk, choose the Status option from the Dbspaces menu. With the cursor on the dbospace that contains the down chunk, press F3 or CTRL-B. The system displays a screen listing all the chunks in the dbospace. Move the cursor to the chunk that is down and type F3 or CTRL-B to recover it.

Recovering a Mirrored Chunk Using the *onspaces* Utility

You can also recover a down chunk using the **onspaces** utility. For example, to recover the chunk that has the pathname **/dev/mirror_chk1** and an offset of 0 kilobytes, you could issue the following command:

```
% onspaces -s db_acct -p /dev/mirror_chk1 -o 0 -O
```

See “onspaces: Modify Blobspaces or Dbspaces” on page 37-40 for a full description of the **onspaces** syntax.

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 mirror chunk before the disk that failed is brought back on-line. Typical UNIX commands to do this 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 mirror 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, **OnLine** immediately releases the mirror 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 Using ON-Monitor

To end mirroring for a dbspace or blobspace using ON-Monitor, select the Mirror option of the Dbspaces menu. Select a dbspace or blobspace that is mirrored and type CTRL-B or F3.

Ending Mirroring Using *onspaces*

You can also end mirroring using the **onspaces** utility. For example, to end mirroring for the root dbspace enter the following command:

```
% onspaces -r rootdbs
```

See “onspaces: Modify Blobspaces or Dbspaces” on page 37-40 for a full description of the **onspaces** syntax.

What Is Data Replication?

Chapter Overview 3

What Is Data Replication? 3

What Is OnLine High Availability Data Replication? 4

What Are Primary and Secondary Database Servers? 4

How Is Data Replication Different from Mirroring? 6

How Is Data Replication Different from Two-Phase Commit? 7

How Does Data Replication Work? 8

How Is the Data Initially Replicated? 8

How Are Updates to the Primary Reproduced on the Secondary? 9

How Are the Log Records Sent? 9

What Are the Data-Replication Buffers? 9

When Are Log Records Sent? 10

Synchronous Updating 10

Asynchronous Updating 11

What Threads Handle Data Replication? 13

Checkpoints Between Database Servers 13

How Is Data Synchronization Tracked? 14

Data-Replication Failures 14

What Are Data-Replication Failures? 14

How Are Data-Replication Failures Detected? 15

What Happens When a Data-Replication Failure is Detected? 15

Administrative Considerations After Data-Replication Failure	16
Actions to Take If the Secondary Database Server Fails	16
Actions to Take if the Primary Database Server Fails	17
Redirection and Connectivity for Data-Replication Clients	19
Designing Clients for Redirection	20
Automatic Redirection: Using DBPATH	20
How Does the DBPATH Redirection Method Work?	20
What Does the Administrator Need to Do?	21
What Does the User Need to Do?	21
Administrator-Controlled Redirection: Changing the <i>sqlhosts</i> File	22
How Does the sqlhosts File Redirection Method Work?	22
What Does the Administrator Need to Do?	22
What Does the User Need to Do?	25
User-Controlled Redirection: INFORMIXSERVER	25
How Does the INFORMIXSERVER Redirection Method Work?	25
What Does the Administrator Need to Do?	26
What Does the User Need to Do?	26
Handling Redirection Within an Application	26
An Example of a Connection Loop and Database Server Type Check	26
Comparison of Different Redirection Mechanisms	28
Designing Clients to Use the Secondary Database Server	29
No Data Modification Statements	29
Locking and Isolation Level	30
Using Temporary Dbspaces for Sorting and Temporary Tables	31

Chapter Overview

This chapter describes **INFORMIX-OnLine Dynamic Server** high availability data replication. The following topics are covered:

- What data replication is, both in a broad sense and in the context of **OnLine**
- How **OnLine** data replication works
- How **OnLine** 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 26, “Using Data Replication,” contains instructions on how to accomplish the administrative tasks involved in using data replication.

What Is Data Replication?

Data replication, in the broadest sense of the term, refers to the process of representing database objects at more than one distinct site.

For example, one way of replicating data is to simply copy a database to a database server installed on a different computer. This allows reports to access the data without disturbing client applications using the original database.

The 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 because 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 for replicated data than for unreplicated data, 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 way 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 client applications) to automatically handle the details of locating and maintaining data replicas.

What Is OnLine High Availability Data Replication?

Within the broad framework of data replication, **OnLine** implements nearly transparent data replication of entire database servers. All of the data managed by one **OnLine** database server is replicated and dynamically updated on another **OnLine** database server, often at a separate geographical location. **OnLine** data replication is sometimes called *high availability* or *hot site backup* because it provides a means of maintaining 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 **OnLine** database servers to use data replication, one **OnLine** 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 not using 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 **OnLine** 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 schematically in Figure 25-1, the secondary **OnLine** is dynamically updated with changes made to the data managed by the primary database server.

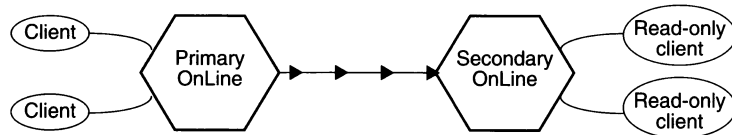


Figure 25-1 *A primary and secondary database server in a data replication pair*

If one of the database servers fails, as shown in Figure 25-2, you can redirect the clients that use that database server to the other database server in the pair.

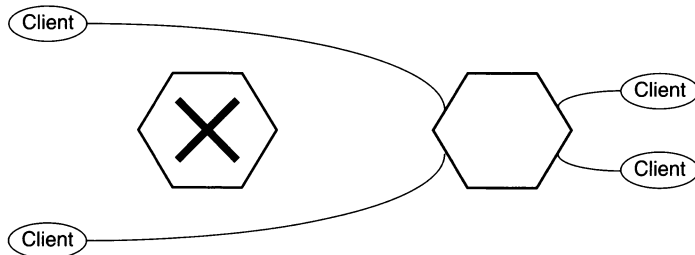


Figure 25-2 *Database servers in a data replication pair and clients after a failure*

If a primary database server fails, you can change the secondary database server to a standard database server so that it can accept updates.

OnLine data replication is designed to:

- Provide for quick recovery if one database server experiences a failure
- Allow for load balancing across the two database servers

How Is Data Replication Different from Mirroring?

OnLine data replication and mirroring are both transparent ways of making **OnLine** more fault-tolerant. However, as shown in Figure 25-3, they are quite different.

Mirroring, described in “What Is Mirroring?” on page 23-3, is the mechanism by which a single **OnLine** 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 **OnLine** automatically updates data on both disks and automatically uses the other disk if one of the dbspaces fails.

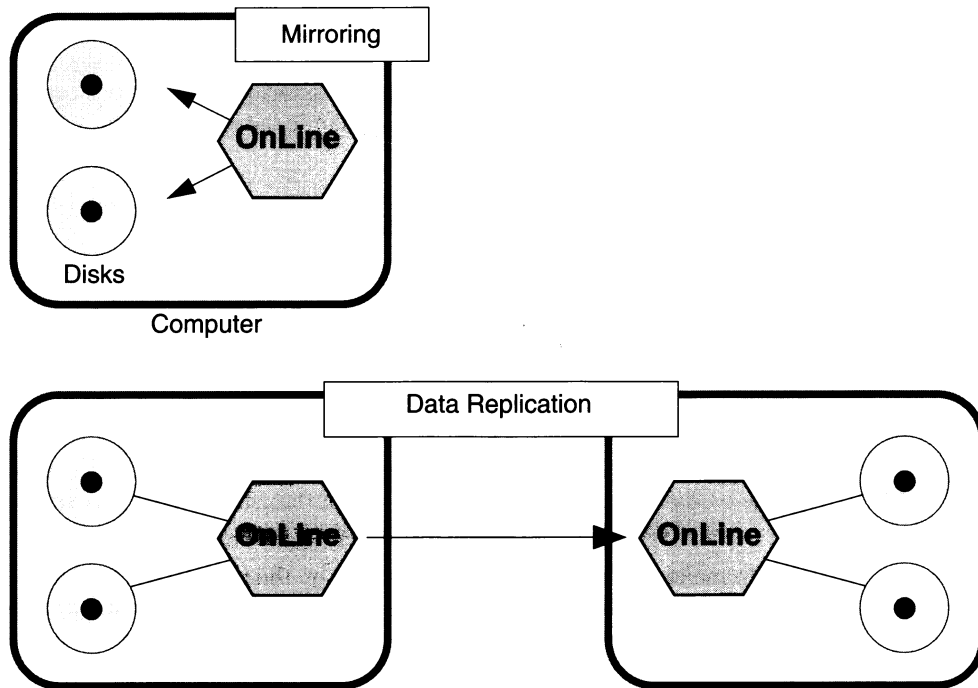


Figure 25-3 A comparison of mirroring and data replication

OnLine 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 **OnLine** data replication involves two separate database servers, the data that these database servers manage is protected against all types of **OnLine** failures—such as a computer crash or the catastrophic failure of an entire site — and not just disk failures.

How Is Data Replication Different from Two-Phase Commit?

The two-phase commit protocol, described in detail in “What Is Two-Phase Commit?” 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 **OnLine** 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 **OnLine** uses to perform data replication. For instructions on how to set-up, start, and administer a data-replication system, refer to Chapter 26, “Using Data Replication.”

How Is the Data Initially Replicated?

OnLine uses archives 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, you create a level-0 archive of all the dbspaces on one database server and restore all the dbspaces from that archive on the other database server in the data-replication pair.
2. Next the database server that you restored from an archive in the first step, reads all the logical-log records generated since that archive from the database server on which the archive was created. The database server first reads the logical-log records 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 26-9.

You must do the initial data replication using an archive. It is not sufficient to use data-migration utilities like **onload** and **onunload** to replicate data because the physical page layout of tables on each database server must be identical for data replication to work.

In the preceding steps, it does not matter whether the database server from which you create the archive is going to be the primary database server or the secondary database server.

When **OnLine** data replication is working, the primary database server is in on-line mode and accepts updates and queries like a standard **OnLine** database server. 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).

How Are Updates to the Primary Reproduced on the Secondary?

OnLine data replication reproduces updates to the primary database server on the secondary database server by having the primary database server send all its logical-log records to the secondary database server as they are generated. (For general information on transaction logging, refer to “What Is Transaction Logging?” on page 16-7.) The secondary database server receives the logical-log records generated on the primary database server and applies them to its dbspaces.

Note: OnLine cannot replicate updates to databases that do not use transaction logging. OnLine does not replicate data in blobspaces either.

How Are the Log Records Sent?

As shown in Figure 25-4 on page 25-10, when the primary database server starts to flush the contents of the logical-log buffer in shared memory to the logical log on disk, **OnLine** 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* (that **OnLine** 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.

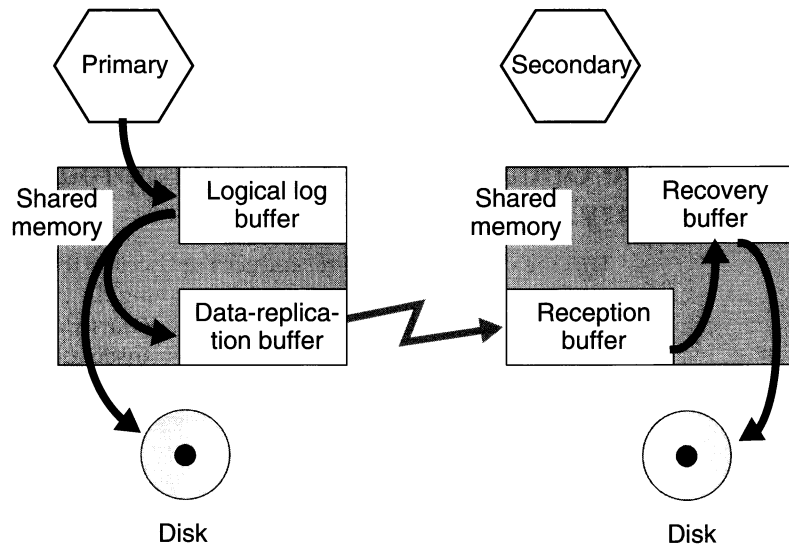


Figure 25-4 How logical-log records are sent from the primary database server to the secondary database server

When Are Log Records Sent?

The primary database server sends the contents of the data-replication buffer to the secondary database server either *synchronously* or *asynchronously*. The value of the ONCONFIG configuration parameter DRINTERVAL, described on page 35-15, determines whether OnLine 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 only completes 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, there is potential for transactions to be lost.

Lost-and-Found Transactions

With asynchronous updating, it is possible that a transaction committed on the primary database server is not 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, it rolls back any open transactions. These transactions include any that were committed on the primary but for which the secondary did not receive a commit record. As a result, there are transactions committed on the primary database server, but not on the secondary database server. When you restart data replication after the failure, **OnLine** 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. This is shown schematically in Figure 25-5.

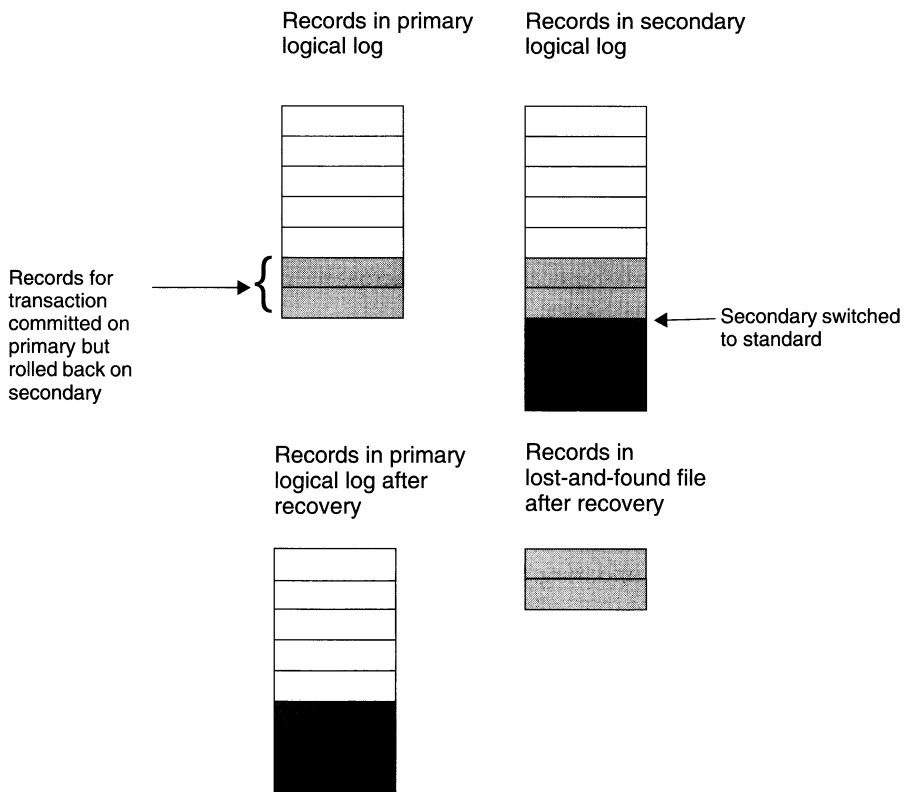


Figure 25-5 Schematic illustrating use of lost-and-found file

If the lost-and-found file appears on the computer running the primary database server after restarting data replication, be aware that a transaction has been lost. **OnLine** 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.

You can reduce the risk of a lost transaction without running data replication in synchronous mode by using unbuffered logging for all the databases. This method reduces the amount of time between the primary database server writing the transaction records to disk and the primary database server sending these records to the secondary database server.

What Threads Handle Data Replication?

OnLine starts specialized threads to support data replication. As shown in Figure 25-6, a thread called **drprsend** on the primary database server sends the contents of the data-replication buffer across the network to a thread called **drsecrcv** on the secondary database server.

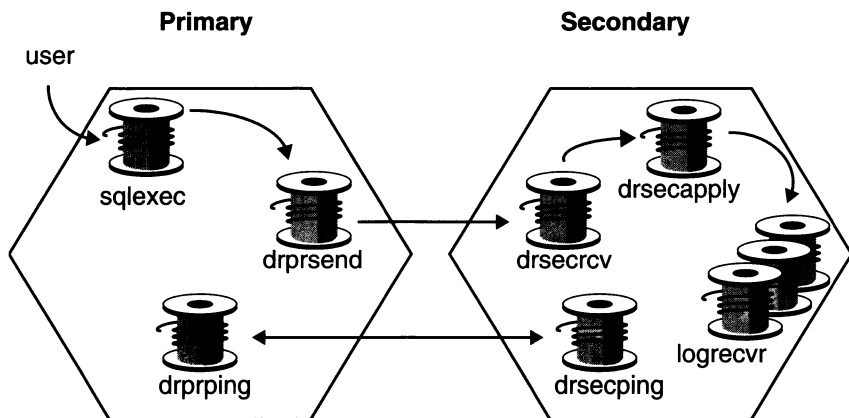


Figure 25-6 Threads that manage data replication

A thread called **drsecapply** on the secondary **OnLine** 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 **OnLine** 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 "OnLine Checkpoints" on page 14-47.) A checkpoint on the primary database server only completes 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 the section “What Are Data-Replication Failures?” which follows.)

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 40-12):

- The id of the logical-log file containing 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 (like a fire or large earthquake) at the site of one of the database servers
- A disruption of the networking cables joining 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, change the type of the database server to standard)
- A disk failure on the secondary database server that is not resolved by a mirrored chunk

Note: 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 back 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 to (or *ping*) 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 or not the primary database server sends any records to the secondary database server. If a database server does not respond to two 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. This means that 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, and the configuration parameter DRAUTO is set to 1, the type of that database server changes automatically to standard. This is explained further under “Actions to Take if the Primary Database Server Fails” on page 25-17.

Administrative Considerations After Data-Replication Failure

You should 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 using the failed database server to *redirect* to the other database server in the pair. How to redirect clients is explained in “Redirection and Connectivity for Data-Replication Clients” on page 25-19.
- How the database servers should react to the failure
Which administrative actions to take after a data-replication failure depend upon whether the primary database server or the secondary database server failed. This is discussed in the following sections: “Actions to Take If the Secondary Database Server Fails” and “Actions to Take if the Primary Database Server Fails”.
If you redirect clients, you also need to consider what sort of load the additional clients will 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. You might also need to increase the USER-THREADS and/or TRANSACTIONS parameters in the ONCONFIG file.

Actions to Take If the Secondary Database Server Fails

If the secondary database server fails, the primary database server remains in on-line mode.

You can redirect clients that used the secondary database server to the primary database server, using any of the methods explained in “Redirection and Connectivity for Data-Replication Clients” on page 25-19. If you redirect these clients, the primary database server might require an additional temporary dbspace for temporary tables and sorting.

You do not need to change the type of the primary database server to standard.

Restarting After the Secondary Database Server Fails

The steps in restarting data replication after a failure of the secondary database server are listed in “Restarting If the Secondary Database Server Fails” on page 26-23.

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 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.

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 only changes the type of the database server. It does not redirect client applications to the secondary database server. You need to redirect clients using any of the mechanisms described in “Redirection and Connectivity for Data-Replication Clients” on page 25-19.

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 below under “Using Automatic Switchover Without a Reliable Network.”

Restarting Data Replication After Automatic Switchover

The steps required to restart data replication after an automatic switchover are listed in “Restarting If the Primary Database Server Fails and Secondary Database Server Is Changed to a Standard Database Server Automatically” on page 26-26.

When the administrator succeeds in bringing the primary database server back up, the type of the secondary database server switches automatically back to secondary, from standard. This switch facilitates the procedure for recovering from failure. The secondary also goes through graceful shutdown to ensure that all clients that might potentially write to the database server are not connected. You need to redirect any clients that perform updates back to the primary database server.

Using Automatic Switchover Without a Reliable Network

Although automatic switchover might sound like a good solution, it is not appropriate for all environments. Consider what would happen if the primary database server did not actually fail, but appeared to fail to the secondary database server. 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 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 both database servers have logical-log records needed by the other. In this situation, you must start from scratch and perform initial data replication with a level-0 archive of one entire database server, as described in “Starting Data Replication for the First Time” on page 26-9.

If your network is not entirely stable therefore, 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 26-17.

Restarting After a Manual Switchover

The steps involved in restarting data replication after a manual switchover are listed in “Restarting If the Primary Database Server Fails and Secondary Database Server Is Changed to a Standard Database Server Manually” on page 26-24.

Restarting If the Secondary Database Server Is Not Switched to Standard OnLine

If the secondary is not changed to type standard either automatically or manually, follow the steps listed in “Restarting If the Primary Database Server Fails and Secondary Database Server Was Not Changed to a Standard Database Server” on page 26-24.

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 also 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.)

OnLine does not have a transparent mechanism for directing client requests to different database servers in a data replication pair, although you can automate this from within the application as described under “Handling Redirection Within an Application” on page 25-26.

Designing Clients for Redirection

In designing 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. There are three different redirection mechanisms:

- Automatic redirection using DBPATH
- Administrator-controlled redirection using the `sqlhosts` file
- User-controlled redirection using INFORMIXSERVER

The mechanism you employ determines which connect syntax you can use in your application. The following three sections each describe one of the redirection mechanisms.

Automatic Redirection: Using DBPATH

This section explains the steps you must follow to redirect clients using 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 using 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 only read (if needed) when an application issues a `CONNECT` statement, applications need to restart for redirection to occur.

An application can contain code which tests to see if a connection has failed and, if so, attempts to reconnect. You do not need to restart such applications for redirection using `DBPATH` to occur.

You can use the following connectivity statement with this method of redirection:

- `CONNECT TO database`

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 is that an application does not use `DBPATH` if a `CONNECT` statement specifies a database server, but only if it specifies a database.

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 which 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 running clients must quit and restart all applications.

Administrator-Controlled Redirection: Changing the *sqlhosts* File

This section explains the steps in redirecting clients using 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 makes a connection to a database server, it finds that database server using information in the **sqlhosts** file.

So, 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?” on page 25-22, 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. In this case 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, “Configuring Connectivity.”

Change the *sqlhosts* File

On the client computer, edit the **sqlhosts** file and 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 specifies information for the database server to which you are redirecting clients in the **nettype**, **hostname**, and **servicename** fields.

Figure 25-7 on page 25-24 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 running the database servers.

Change Other Connectivity Files

You also must ensure that the following statements are true on the client computer before that client can reconnect to the other database server:

- The **/etc/hosts** file has an entry for the hostname of the computer running the database server to which you are redirecting clients.
- The **/etc/services** file has an entry for the **servicename** of the database server to which you are redirecting clients.

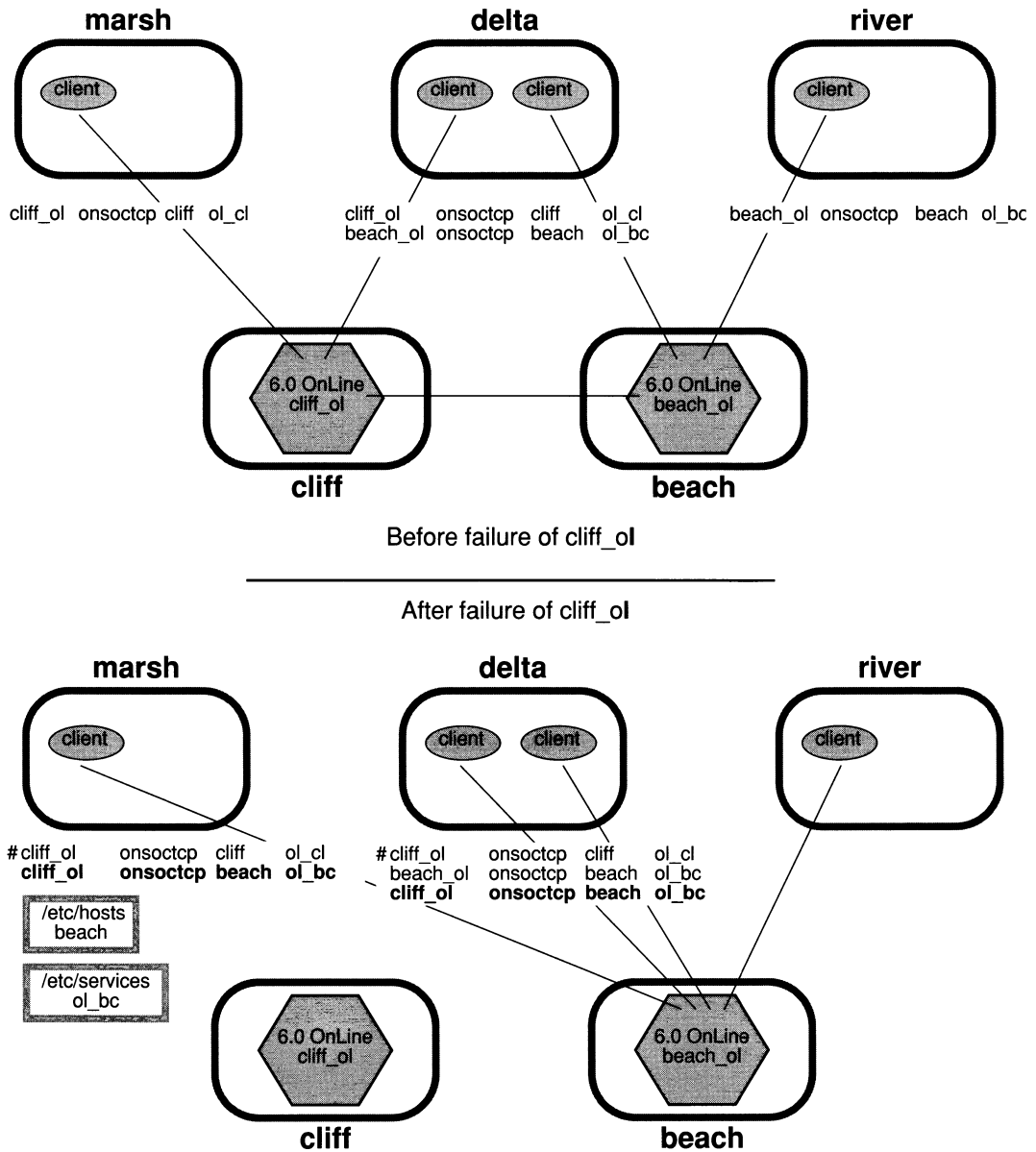


Figure 25-7 The sqlhosts file entries before and after a failure of the cliff_ol database server

What Does the User Need to Do?

After the administrator makes these changes, clients connect to the database server the administrator redirects them to when they issue their next CONNECT statement.

If your applications contain code which 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 running clients must quit and restart all applications.

User-Controlled Redirection: INFORMIXSERVER

This section explains the steps in redirecting clients using 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 using that database server can reset their INFORMIXSERVER environment variable to the other database server in the pair to access the same data.

Applications only read the value of the INFORMIXSERVER environment variable when they start. Applications must be restarted, therefore, 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 because, 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. Administrators might need to attend to the type of the database server, however.

What Does the User Need to Do?

Users running client applications must perform the following three steps when they decide to redirect clients using 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 which tries repeatedly to connect with the other database server in the pair. This routine redirects clients without the user having to exit the application and restart it.

An Example of a Connection Loop and Database Server Type Check

Figure 25-8 on page 25-27 shows an example of a function in a client 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 server is still a secondary, it calls another function to alert the user (or OnLine 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 "stores6";
        if (strcmp(SQLSTATE, "00000"))
        {
            if (sqlca.sqlwarn.sqlwarn6 != 'W')
            {
                notify_admin();
                if (tries < MAXTRIES - 1)
                    sleep(15);
            }
            else connected = 1;
        }
    }
    return ((tries == MAXTRIES)? -1:0);
}

```

Figure 25-8 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 were to use the `sqlhosts` file redirection method (explained in "Administrator-Controlled Redirection: Changing the `sqlhosts` File" on page 25-22), you might have a different connection statement, as follows:

```
EXEC SQL CONNECT TO "stores6@cliff_ol";
```

In this example, `stores6@cliff_ol` refers to a database on a database server that is recognized by the client's computer. In order 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 the client waits before retrying to connect or the number of tries the function makes. You should 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 25-9 summarizes the differences between the three redirection mechanisms.

	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 <code>sqlhosts</code> 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 <code>INFORMIXSERVER</code> 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 using a given database server are redirected.	Individual clients are redirected.	Individual clients are redirected.
Are changes to environment variables required?	No		No		Yes

Figure 25-9 Comparison of redirection methods for different connectivity strategies

Designing Clients to Use the Secondary Database Server

You can achieve a degree of load balancing when using data replication by having some client applications use the secondary database server in a data-replication pair. You must design all client applications that use the secondary database server with the following points in mind:

- Any statements which attempt to modify data fail.
- Locking and isolation levels are not the same as on standard **OnLine** 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 which are in logical recovery (which includes all dbspaces on the secondary database server) are not allowed. For example, the following statements produce errors:

- ALTER INDEX
- ALTER TABLE
- CREATE DATABASE
- CREATE INDEX
- CREATE PROCEDURE
- CREATE PROCEDURE FROM
- CREATE SCHEMA
- CREATE SYNONYM
- CREATE TABLE
- CREATE VIEW
- DELETE
- DROP DATABASE
- DROP INDEX
- DROP PROCEDURE
- DROP SYNONYM
- DROP TABLE
- DROP VIEW
- GRANT
- INSERT

- LOAD
- RENAME COLUMN
- RENAME TABLE
- REVOKE
- UNLOAD
- UPDATE
- UPDATE STATISTICS

To prevent clients 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 updating statements, you can use the fact that on the secondary database server, **OnLine** sets the **sqlwarn6** of the **sqlwarn** field in the **ESQL/C sqlca** structure (and equivalent values for other SQL APIs) to **W**.

Locking and Isolation Level

Because all clients using the secondary database server only read data, locking to ensure isolation between those clients is not required. However, a client using the secondary database server is not protected from the activity of users on the primary database server, because the **logrecvr** threads performing 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 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 occurring on the primary database server, even though a client using 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 10-18 explains where **OnLine** finds temporary space to use during a sort or for a temporary table. To prevent the secondary database server from writing to a dbspace which is in logical-recovery mode, you must take one (or all) of the following actions:

- Ensure that a temporary dbspace exists. (See “Creating a Dbspace” on page 11-8 for instructions on creating a temporary dbspace.)
- 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

Chapter Overview	3
Planning for Data Replication	3
Configuring Data Replication	4
Meeting Hardware and Operating-System Requirements	4
Meeting Database and Data Requirements	5
Meeting Database Server Configuration Requirements	5
Version	5
DbSpace and Chunk Configuration	6
Mirroring	6
Physical-Log Configuration	6
System Archive and Logical-Log Tape Devices	6
Logical-Log Configuration	7
Shared-Memory Configuration	7
Data-Replication Parameters	7
Configuring Data-Replication Connectivity	8
Starting Data Replication for the First Time	9
Performing Basic OnLine Administration Tasks	12
Changing Database Server Configuration Parameters	12
Archiving and Logical-Log File Backups	12
Changing the Logging Status of Databases	13
Adding and Dropping Chunks, DbSpaces, and BlobSpaces	13
Using and Changing Mirroring of Chunks	13
Managing the Physical Log	14

Managing the Logical Log	14
Managing Virtual Processors	15
Managing Shared Memory	15
Changing the Database Server Mode	15
Changing the Database Server Type	16
Changing the Database Server Type of the Primary Database Server	17
Changing the Database Server Type of the Secondary Database Server	17
Restoring Data If Media Failure Occurs	18
Restoring After Media Failure on the Primary	18
Restoring After Media Failure on Secondary Database Server	19
Restarting Data Replication After a Failure	20
Restarting After Critical Data Is Damaged	20
Critical Media Failure on the Primary Database Server	21
Critical Media Failure on the Secondary Database Server	22
Critical Media Failure on Both Database Servers	22
Restarting If Critical Data Is Not Damaged	23
Restarting If the Secondary Database Server Fails	23
Restarting If the Primary Database Server Fails and Secondary Database Server Was Not Changed to a Standard Database Server	24
Restarting If the Primary Database Server Fails and Secondary Database Server Is Changed to a Standard Database Server Manually	24
Restarting If the Primary Database Server Fails and Secondary Database Server Is Changed to a Standard Database Server Automatically	26

Chapter Overview

This chapter describes how to use **INFORMIX-OnLine Dynamic 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 **OnLine** 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 25, “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 **OnLine** database server to store data that you wish to replicate, migrate and redistribute this data so that it can be managed by a single database server.
- Ensure that all databases that you want to replicate use transaction logging. To turn on transaction logging, see Chapter 17, “Managing Database Logging Status.”

- Develop client applications to make use of both database servers in the data-replication pair. Read “Redirection and Connectivity for Data-Replication Clients” on page 25-19 and “Designing Clients to Use the Secondary Database Server” on page 25-29 for a discussion of design considerations.
- Create a schedule for starting data replication for the first time.
- Design an archive 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 25-19.

Configuring Data Replication

To configure your system for data replication you must

- 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 an **OnLine** data-replication database server pair to function, it must meet the following hardware requirements:

- The computers running the primary and secondary **OnLine** database servers must be identical (same vendor and architecture).
- The operating systems on the computers running the primary and secondary **OnLine** database servers must be identical.
- The hardware running the primary and secondary **OnLine** database servers must support network capabilities.
- The amount of disk space allocated to nontemporary dbspaces for the primary and secondary **OnLine** 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.

It is also important for the **OnLine** administrators of both the database servers to be able to communicate when performing administrative tasks (for example by telephone).

Meeting Database and Data Requirements

For an **OnLine** 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 **OnLine** database server uses logical-log records from the primary **OnLine** 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, leaving the secondary database server with 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 Using ON-Archive” on page 17-5 or “Turning on Transaction Logging Using ontape” on page 17-7.

- If your primary database server has blobs stored in blobspaces, modifications to the data within those blobspaces is not replicated as part of normal data-replication processing. Blob 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 **OnLine** database server configuration requirements.

Meeting these requirements requires that you fully configure each of the **OnLine** database servers. Refer to “Configuring a Production Environment” on page 3-17 for information on configuring a database server. You can then use the relevant aspects of that configuration to configure the other database server in the pair.

Version

The versions of **OnLine** on the primary and secondary database servers must be version 6.0 or greater and 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 **OnLine** database servers.

The configuration must contain at least one temporary dbospace. See “Using Temporary Dbospaces for Sorting and Temporary Tables” on page 25-31.

You can use symbolic links for the chunk pathnames, as explained in “Creating Links to Each Raw Device” on page 11-6.

The following ONCONFIG parameters must have the same value on each database server:

- ROOTNAME (see page 35-35)
- ROOTPATH (see page 35-36)
- ROOTOFFSET (see page 35-35)
- ROOTSIZE (see page 35-36)

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 mirror chunk for the root chunk of the primary database server, however, you must also specify a mirror chunk for the root chunk on the secondary database server. Therefore, the following ONCONFIG parameters must be set to the same value on both database servers:

- MIRRORPATH (see page 35-26)
- MIRROROFFSET (see page 35-26)

Physical-Log Configuration

The physical log should be identical on both servers. The following ONCONFIG parameters must have the same value on each database server:

- PHYSDBS (see page 35-33)
- PHYSFILE (see page 35-33)

System Archive and Logical-Log Tape Devices

You can specify different tape devices for the primary and secondary database servers.

The tape size and tape block size for the archive and logical-log tape devices should be identical. The following ONCONFIG parameters must have the same value on each database server:

- TAPEBLK (see page 35-41)
- TAPESIZE (see page 35-44)
- LTAPEBLK (see page 35-23)
- LTAPESIZE (see page 35-24)

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 35-20)
- LOGSIZE (see page 35-21)

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 35-15)
- DRLOSTFOUND (see page 35-15)
- DRTIMEOUT (see page 35-16)
- DRAUTO (see page 35-14)

Configuring Data-Replication Connectivity

For an OnLine 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 running OnLine in a data-replication pair must have at least the following entries:

- An entry identifying the OnLine database server running on that computer
- An entry identifying the other OnLine database server in the data-replication pair

Figure 26-1 shows a sample data-replication configuration and example `sqlhosts` file entries necessary for data replication.

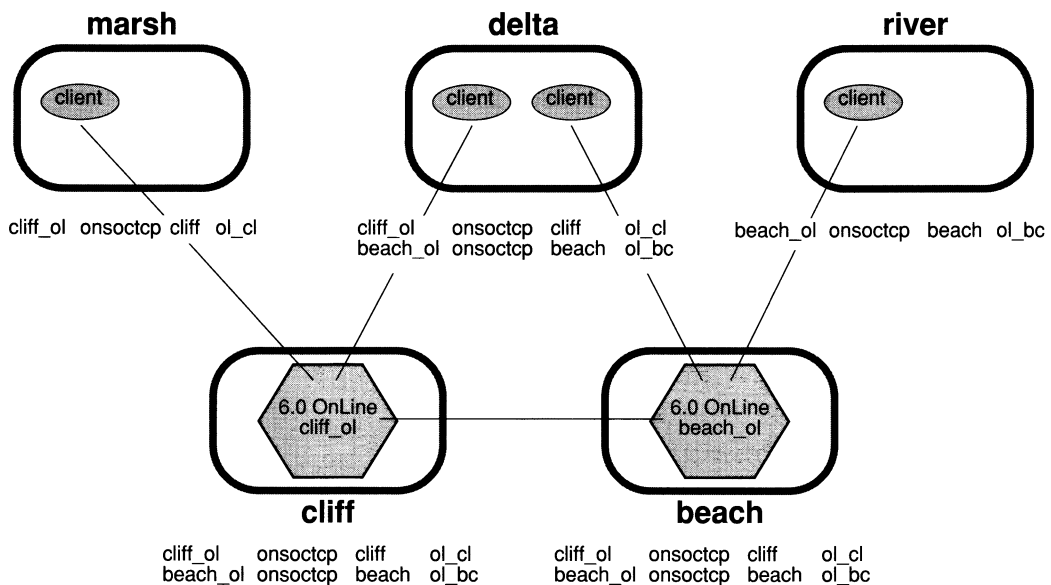


Figure 26-1 Example `sqlhosts` file entries for database servers in a data replication pair

In addition to entries in the `sqlhosts` files, the computers running OnLine 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 26-2 on page 26-10 lists the commands required to perform each step. You can perform some of the steps using either the ON-Archive or the **ontape** utility. In such cases, the ON-Archive command and the equivalent **ontape** command are both indicated. You must employ the same utility throughout the procedure, however. Figure 26-2 also shows messages sent to the message log.

1. Create a level-0 archive 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 only succeeds 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 archive 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 of these logical-log records still reside on the primary database server disk, the primary 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 there are logical-log records written to the primary database server that 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 running the primary database server, or to a device at the same location as the primary database server.

After recovering all the logical-log files on tape, the logical restore completes using the logical-log files on the primary database server disk.

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 secondary Messages to message log DR: new type = primary, secondary server name = secondary DR: trying to connect to secondary server DR: Cannot connect to secondary server	
3		ON-Archive command ONDATARTR> RETRIEVE/DBSPACESET=*/REQUEST=rid/TAPE =(primary:/dev/remotedrive) ontape command % ontape -p Answer no when prompted to back up the logs.

Figure 26-2 Steps in starting data replication for the first time, with explanations

Step	On the Primary	On the Secondary
		<p>Messages to message log INFORMIX-OnLine Initialized -- Shared Memory Initialized Recovery Mode Physical restore of rootdbs started. Physical restore of rootdbs Completed.</p>
4	<p>Messages to message log DR: Primary server connected DR: Primary server operational</p>	<p>onmode command % onmode -d secondary <i>primary</i></p> <p>Messages to message log DR: new type = secondary, primary server name = <i>primary</i></p> <p>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).</p> <p>Messages to message log DR: Trying to connect to primary server DR: Secondary server connected DR: Failure recovery from disk in process. Logical Recovery allocating n worker threads ('OFF_RECVR_THREADS'). Logical Recovery Started Start Logical Recovery - Start Log n, End Log? Starting Log Position - n 0xnxxxx DR: Secondary server operational</p>
5	<p>Messages to message log DR: Primary server connected DR: Primary server operational</p>	<p>ON-Archive command ONDATARTR> RETRIEVE/LOGFILE/TAPE=(primary:/dev/re motedevice)</p> <p>ontape command % ontape -l</p> <p>Messages to message log 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 0xnxxxx DR: Secondary server operational</p>

Figure 26-2 Steps in starting data replication for the first time, with explanations

Performing Basic OnLine Administration Tasks

This section contains instructions on how to perform basic **OnLine** administration tasks once your system is running data replication.

Changing Database Server Configuration Parameters

Some of the **OnLine** configuration parameters must be set to the same value on both database servers in the data-replication pair. Other **OnLine** 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 using the **onmode -k** option. If DRAUTO is set to 1, it is easiest if you 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 on-line first. Figure 26-3 on page 26-16 and Figure 26-4 on page 26-16 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.

Archiving and Logical-Log File Backups

When using data replication, you must back up logical-log files and create archives of your data, just as you would with a standard **OnLine** database server. You only need to perform archives and logical-log file backups on the primary database server. Be prepared, however, to perform archives 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 archiving and logical-log backup tool (either ON-Archive or **ontape**) on both database servers. You can, however, change tools at the point of a level-0 archive. So, for example, you might do the initial

set up described in “Starting Data Replication for the First Time” on page 26-9 with **ontape**, but then use ON-Archive for performing regular archives.

The block size and tape size used (for both archiving and logical-log backups) must be identical on the primary and secondary **OnLine** database servers.

Changing the Logging Status of Databases

You cannot add transaction logging to databases on the primary database server while 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 an archive and restore as described in “Starting Data Replication for the First Time” on page 26-9.

Adding and Dropping Chunks, Dbspaces, and Blobspaces

You can only perform disk-layout operations such as adding or dropping chunks, dbspaces, and blobspaces, from the primary database server. The operation is replicated on the secondary database server. This 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.

You can only perform disk-layout operations from the primary database server, however. This means that you can only add or drop a mirror chunk from the primary database server. A mirror chunk that you add or drop to the primary database server is added or dropped to the secondary database

server as well. Even if you only want to mirror a dbspace on one of the database servers in the data-replication pair, you must create mirror chunks for that dbspace on *both* database servers.

Before you can add a mirror chunk, the disk space for that chunk must already be allocated on both the primary and secondary database servers. See “Allocating Disk Space” on page 11-3 for general information on allocating disk space.

You can take a mirror chunk down or recover a mirror chunk on either the primary or secondary database server. These processes are transparent to data replication.

Managing the Physical Log

If you make changes to the physical log on one database server, you must make the same changes at the same time to the physical log on the other database server. See “Changing Database Server Configuration Parameters” on page 26-12 for the procedure to follow for making this change.

Managing the Logical Log

The size of the logical log must be the same on both database servers. If you add or drop a logical-log file on one database server, you must make the same change to the other database server in the pair.

You can add or drop a logical-log file using the **onparams** utility, as described in Chapter 19, “Managing Logical-Log Files.” **OnLine** replicates this change on the secondary database server; however, the LOGFILES parameter on the secondary database server is not updated. After issuing the **onparams** command from the primary database server, therefore, you must manually change the LOGFILES parameter to the desired value. Finally, for the change to take effect you must perform a level-0 archive 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 archive. The new logical-log file on the secondary database server is still flagged “A”. However, this 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. See “Changing Database Server Configuration Parameters” on page 26-12 for the procedure to follow for making this change.

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 26-3 summarizes the effects of changing the mode of 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; mode remains read-only. If DRAUTO is set to 1, secondary switches to standard type and can accept updates.	Treat it like a failure of the primary. There are three different scenarios, depending on what you do with the secondary while the primary is off-line: <ul style="list-style-type: none"> • “Restarting If the Primary Database Server Fails and Secondary Database Server Was Not Changed to a Standard Database Server” on page 26-24 • “Restarting If the Primary Database Server Fails and Secondary Database Server Is Changed to a Standard Database Server Manually” on page 26-24 • “Restarting If the Primary Database Server Fails and Secondary Database Server Is Changed to a Standard Database Server Automatically” on page 26-26
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 26-3 Mode changes on the primary database server

Figure 26-4 summarizes the effects of changing the mode of 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: <ul style="list-style-type: none"> • “Restarting If the Secondary Database Server Fails” on page 26-23

Figure 26-4 Mode changes on the secondary database server

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 26-15). 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 you use to change the database server type is **onmode**. Reference information for **onmode** is in “onmode: Mode and Shared-Memory Changes” on page 37-27.

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 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, 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 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 26-9.

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

Figure 26-5 on page 26-19 summarizes the various scenarios for restoring data if the primary database server suffers media failure. Note the following issues:

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 archives (or the secondary archives if the secondary database server was switched to standard mode and activity redirected). See “Restarting After Critical Data Is Damaged” on page 26-20.

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 an archive and next a logical restore of all logical-log records written since that archive. (See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for more

information.) You must back up all logical-log files before you perform the warm restore.

Data-Replication Server	Critical Media?	Chunks Mirrored?	Effect of Failure and Procedure for Restoring Media
Primary	Yes	No	Primary database server fails. Follow the procedure in "Restarting After Critical Data Is Damaged" on page 26-20.
Primary	Yes	Yes	Primary database server remains on-line. Follow the procedures in "Recovering a Mirrored Chunk" on page 24-9.
Primary	No	No	Primary database server remains on-line. Follow procedure in the <i>INFORMIX-OnLine Dynamic Server Archive and Backup Guide</i> for performing a warm restore of a dbspace from an archive. Back up all logical-log files before performing the warm restore.
Primary	No	Yes	Primary database server remains on-line. Follow the procedures in "Recovering a Mirrored Chunk" on page 24-9.

Figure 26-5 Different scenarios for media failure on the primary database server

Restoring After Media Failure on Secondary Database Server

Figure 26-6 on page 26-20 summarizes the various scenarios for restoring data if the secondary database server suffers media failure. Note the following issues:

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 archives on the primary database server. (See "Restarting After Critical Data Is Damaged" on page 26-20.) In the second case, you cannot restore selected dbspaces from the secondary archive because

they might now deviate from the corresponding dbspaces on the primary database server. You must do a full restore.

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 26-20
Secondary	Yes	Yes	Secondary database server remains on-line in read-only mode. Follow the procedures in "Recovering a Mirrored Chunk" on page 24-9.
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 26-20.
Secondary	No	Yes	Secondary database server remains on-line in read-only mode. Follow the procedures in "Recovering a Mirrored Chunk" on page 24-9.

Figure 26-6 Different scenarios for media failure on the secondary database server

Restarting Data Replication After a Failure

"What Are Data-Replication Failures?" on page 25-14 discusses the various types of data-replication failure. The procedure you must follow to restart data replication depends on whether or not critical data was damaged on one of the database servers. Both cases are discussed in this section.

Restarting After Critical Data Is Damaged

If one of the database servers experiences a failure that damages the root dbspace, the dbspace containing logical-log files, or the dbspace containing 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 26-7 lists the commands required to perform this procedure:

1. If the secondary database server was changed to a standard database server manually, bring this database server to quiescent mode and then use the **onmode -d** command to change the type back to secondary.
If the secondary database server was changed to a standard database server automatically (DRAUTO = 1), this step does not apply. The type of the database server is automatically changed back to secondary when you bring the primary database server back on-line.
2. Restore the primary database server from the last archive.
3. Use the **onmode -d** command to set the type of the primary database server and 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 of the primary database server cannot complete because you backed-up and freed logical-log files on the 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.

Step	On the Primary	On the Secondary
1		onmode command % onmode -s % onmode -d secondary primary
2	ontape command % ontape -p ON-Archive command ONDATARTR> RETRIEVE/DBSPACESET=*/REQUEST=rid /TAPE=(primary:/dev/remotedrive)	
3	onmode command % onmode -d primary secondary	
4	ontape command % ontape -l ON-Archive command ONDATARTR> RETRIEVE/LOGFILE/TAPE=(secondary: /dev/remotedevice)	

Figure 26-7 Steps for restarting data replication after a critical media failure on the primary database server

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 26-9.

Critical Media Failure on Both Database Servers

In the unfortunate event that both of the machines running database servers in a data replication pair experience a failure that damages the root dbspace, the dbspace containing logical-log files, or the dbspace containing the physical log, perform the following tasks to restart data replication:

1. Restore one database server—it does not matter which one—from archive and logical-log backup tapes.
2. After restoring 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 26-9). Use the functioning database server with the intact disk(s) as the database server with the data.

Restarting If Critical Data Is Not Damaged

When there has not been any damage to critical data on either database server, the following four scenarios, each requiring different procedures for restarting data replication, are possible:

- 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).

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 26-8. The steps assume that you have been backing up logical-log files on the primary database server as necessary since the failure of the secondary database server.

Step	On the Primary	On the Secondary
1	The primary database server should be in on-line mode.	<pre>% oninit If you receive the following message in the message log, continue with step 2. DR: Start Failure recovery from tape</pre>
2		<pre>ON-Archive command Onarchive> CATALOG/VSET=remote_logs/VOLUME=volnum /SID=sysid Onarchive>RETRIEVE/LOGFILE/VSET=remote_logs ontape command % ontape -l</pre>

Figure 26-8 Steps in restarting after a failure on the secondary database server

Restarting If the Primary Database Server Fails and 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 changed to a standard database server, simply bring the primary database server back on-line using **oninit**.

Restarting If the Primary Database Server Fails and 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 26-9 on page 26-25.

Step	On the Primary	On the Secondary
1		<pre>% onmode -s</pre> <p>This step takes the secondary database server (now a standard) to quiescent mode. All clients that are connected to this server will have to disconnect. Applications that perform updates must be redirected to the primary. See "Redirection and Connectivity for Data-Replication Clients" on page 25-19.</p>
2		<pre>% onmode -d secondary primary</pre>
3	<pre>% oninit</pre> <p>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 there are logical-log files on the secondary that you have backed-up and freed, 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).</p> <p>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 running the secondary database server.</p> <p>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.</p>	
4	<p>If you are prompted to recover logical-log records from tape perform this step.</p> <p>ON-Archive command <pre>Onarchive> CATALOG/VSET=remote_logs/VOLUME= volnum/SID=sysid Onarchive>RETRIEVE/LOGFILE/VSET= remote_logs</pre> </p> <p>ontape command <pre>% ontape -l</pre> </p>	

Figure 26-9 Steps for restarting after a failure on the primary database server and secondary database server was changed to a standard database server manually

Restarting If the Primary Database Server Fails and 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 25-17), complete the steps in Figure 26-10.

Step	On the Primary	On the Secondary
1	<p><code>% oninit</code></p> <p>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.</p> <p>If there are logical-log files on the secondary that you have backed-up and freed, 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).</p> <p>For ontape users: Set the logical-log tape device to a device on the computer running the secondary database server.</p> <p>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.</p>	<p>The secondary database server automatically goes through graceful shutdown when you bring the primary back up. This ensures that all clients are disconnected. Any applications that perform updates must be redirected back to the primary database server. See “Redirection and Connectivity for Data-Replication Clients” on page 25-19.</p>
2	<p>If you are prompted to recover logical-log records from tape, perform this step.</p> <p>ON-Archive command</p> <pre>Onarchive> CATALOG/VSET=remote_logs/VOLUME=volnum/S ID=sysid Onarchive>RETRIEVE/LOGFILE/VSET=remote_1 ogs</pre> <p>ontape command</p> <pre>% ontape -l</pre>	

Figure 26-10 Steps in restarting after a failure on the primary server and secondary was changed to a standard database server manually

What Is Consistency Checking?

Chapter Overview	3
Performing Periodic Consistency Checking	3
Verify Consistency	4
oncheck -cr	4
oncheck -cc	4
oncheck -ce	5
oncheck -cl	5
oncheck -cD	5
Monitor for Data Inconsistency	6
Retain Consistent Level-0 Archive	7
Dealing with Corruption	7
Symptoms of Corruption	8
Run oncheck First	8
I/O Errors on a Chunk	8
Collecting Diagnostic Information	9

Chapter Overview

INFORMIX-OnLine Dynamic Server is designed in such a way that it detects problems that might be caused by hardware or operating system errors or by unknown problems within **OnLine**. It detects problems by performing *assertions* in many of its critical functions. An assertion is merely 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, **OnLine** reports an *assertion failure*. Some text describing the check that failed is recorded in the **OnLine** message log. **OnLine** 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 has sections about 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 archives, Informix recommends that you periodically take the following actions:

- Verify that all data and **OnLine** overhead information is consistent
- Check the message log for assertion failures while you verify consistency
- Create a level-0 archive after consistency is verified

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 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 archive.

Run the following commands as part of the check:

- **oncheck -cr**
- **oncheck -cc**
- **oncheck -ce**
- **oncheck -cI *dbname***
- **oncheck -cD *dbname***

The following sections describe these commands.

You can run each of these commands while **OnLine** is in on-line mode. See “Locking and oncheck” on page 37-7 for information on how **oncheck** locks objects as it checks them and which users can run **oncheck**.

In most cases, if one or more of these checks detects an error, the solution is to restore the database from an archive. However, the source of the error might also be your hardware or operating system.

oncheck -cr

Execute **oncheck -cr** to validate the **OnLine** reserved pages that reside at the beginning of the initial chunk of the root dbspace. These pages contain the primary **OnLine** overhead information. If this command detects errors perform a data restore from archive. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for instructions on how to restore your data from an archive.

This command might report warnings. In most cases, these warnings call your attention to situations you are already aware of.

oncheck -cc

Execute **oncheck -cc** to validate the system catalog tables for each of the databases that **OnLine** 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 an error message is returned from **oncheck -cc**, the situation is quite different. You should contact Informix Technical Support immediately.

oncheck -ce

Execute **oncheck -ce** to validate the extents in every **OnLine** database. It is important that extents do not overlap. If this command detects errors, perform a data restore from archive. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for instructions on how to restore your data from an archive.

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. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for instructions on how to restore your data from an archive.

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. See Chapter 31, "Data Migration." If this does not succeed, perform a data restore from archive. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for instructions on how to restore your data from an archive.

Monitor for Data Inconsistency

If the consistency-checking code detects an inconsistency during **OnLine** operation, an assertion failure is reported to the **OnLine** message log. (See “What Is the Message Log?” on page 29-6.)

Assertion failures take the following form 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 OnLine entity*
Action: *What action the OnLine administrator should take*
See Also: *file(s) containing additional diagnostics*

The “See Also:” line contains one or more of the following filenames:

- */pathname/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 **OnLine** shared memory at the time of the assertion failure but only if the ONCONFIG parameter DUMP-SHMEM is set to 1.

The file **gcore.xxx** contains a core dump of the **OnLine** 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 **OnLine** 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 **OnLine** 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, **OnLine** stops immediately when an assertion fails. However, when failures appear specific to a table or smaller entity, the **OnLine** continues to run.

When an assertion fails because of inconsistencies on a data page that **OnLine** 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 38, “OnLine Message Log Messages,” which describes the messages that can appear in the **OnLine** message log, provides additional details about the objectives and contents of messages.

Retain Consistent Level-0 Archive

After you perform the checks described in “Verify Consistency” on page 27-4 without errors, create a level-0 archive. Retain this archive 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 archive. However if you do not, then at minimum, keep all the tapes necessary to recover from the archive that was created immediately after **OnLine** was verified to be consistent.

Dealing with Corruption

This section describes some of the symptoms of **OnLine** system corruption and actions that **OnLine** or you, as administrator, can take to resolve the problems. Corruption in an **OnLine** database can occur as a consequence of problems caused by hardware or the operating system, or from some unknown **OnLine** problems. Corruption can affect either data or **OnLine** overhead information.

Symptoms of Corruption

OnLine 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 **OnLine** message log. They always indicate files that contain additional diagnostic information that can help you determine the source of the problem. (See “Monitor for Data Inconsistency” on page 27-6.)
- 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 **OnLine** 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 (**OnLine** 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** is unable to find or access the table or database, perform the overhead checks described in, “Verify Consistency” on page 27-4.

I/O Errors on a Chunk

If an I/O error occurs during **OnLine** operation, the status of the chunk on which the error occurred changes to down. If a chunk is down, the **onstat -d** display shows the chunk status as PD- for a primary chunk and MD- for a

mirror chunk. A message written to the **OnLine** 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, **OnLine** continues to operate using the mirror chunk. Use operating-system utilities to determine what is wrong with the down chunk and then to correct the problem. You must then direct **OnLine** to restore mirrored chunk data. See "Recovering a Mirrored Chunk" on page 24-9 for instructions on how to recover a mirrored chunk.

If the down chunk is not mirrored and contains logical-log files, the physical log, or the root dbspace, **OnLine** immediately initiates an abort. Otherwise, **OnLine** can continue to operate, but cannot write to or read from the down chunk or any other chunks in that chunks dbspace. You must take steps first to determine why the I/O error occurred, then correct the problem and, finally, restore the dbspace from an archive.

If you take **OnLine** to off-line mode when a chunk is marked as down ("D"), you can reinitialize **OnLine** provided the chunk marked down does not contain critical media (logical-log files, the physical log, or the root dbspace).

Collecting Diagnostic Information

OnLine facilitates the collection of diagnostic information through the setting of several ONCONFIG parameters. Since an assertion failure is generally an indication of an unforeseen problem, whenever one occurs, you should notify Informix Technical Support. 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 is 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. You will probably want to fax or email the *af.xxx* file to Informix technical support because it often contains the information they need to resolve the problem.

The following ONCONFIG parameters direct **OnLine** to preserve diagnostic information whenever an assertion failure is detected or whenever **OnLine** enters into an abort sequence:

- DUMPCORE page 35-17
- DUMPGCORE page 35-18
- DUMPSHMEM page 35-18
- DUMPDIR page 35-17
- DUMPCNT page 35-16

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.

Note: 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 only useful if this is the case.

OnLine 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

Chapter Overview 3

Situations to Avoid in Administering OnLine 3

Chapter Overview

Occasionally, **INFORMIX-OnLine Dynamic Server** administrators conceive of a shortcut that seems like a good idea . Because of the complexity of **OnLine**, an idea that appears to be an efficient time saver can create problems elsewhere during operation. This chapter attempts to safeguard you from bad ideas that sound good.

The *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* has advice on situations to avoid for archiving.

Situations to Avoid in Administering OnLine

The following ideas might sound good in theory, but they have unexpected consequences that could adversely affect your **OnLine** performance.

- Never make changes to the tables in the **sysmaster** database.
- Never kill an **OnLine** process (virtual processor) because it will cause **OnLine** 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 servicename. See “The \$INFORMIXDIR/etc/sqlhosts File” on page 4-10 for information on the servicename.
- Avoid transactions that span a significant percentage of available logical-log space. See the descriptions of the LTXHWM and LTXEHWM parameters in Chapter 35, “OnLine Configuration Parameters.”
- Do not rely on **dbexport** (a utility that creates a copy of your database for migrating) as an alternative to creating routine archives.
- Do not run utilities that send output to tape in background mode (using the **&** operator).
- Do not move a chunk from one dbspace or blobspace to another without doing a level-0 archive of both spaces (that is, the *before* space and the *after*

space). If you do not archive using level-0, a potential problem exists if the two spaces involved are restored in parallel.

- Do not locate mirror chunks on the same device as the primary chunks. Ideally, place the mirror chunks on devices that are managed by a different controller than the one that manages the primary chunks.

Index

Boldface page numbers indicate the main discussion of a topic.
Special symbols are listed in ASCII order at the end of the index.

A

Adding virtual processors 12-9, 12-14

ADM (Administration virtual processor) 12-30

Administrative tasks

- before installation 3-7
- configuration tasks 2-4
- consistency checking 27-3
- controlling location of storage 10-15
- cron jobs 3-29
- for production environment 3-17
- in quiescent mode 7-3
- initial tasks 2-3
- list of tasks 3-28
- list of tools 3-5
- managing the sqlhosts file 4-7
- network configuration files 4-7
- of database administrator 1-10
- of OnLine administrator 1-10
- of OnLine operator 1-10
- planning 3-4
- routine tasks 2-3
- situations to avoid 28-3
- startup and shutdown scripts 3-28
- types of 2-3

Database administrator

See also Administrative tasks.

ADT (audit) virtual processor 12-31

ADTERR parameter, description of 35-6

ADTMODE parameter

description of 35-7

mentioned 12-31

ADTSIZE parameter, description of 35-7

AFF_NPROCS parameter
 description of 35-8
 purpose of 12-18
 recommendation for 30-19
 with MULTIPROCESSOR parameter 35-27

AFF_SPROC parameter
 description of 35-8
 purpose of 12-18
 with MULTIPROCESSOR parameter 35-27

AIO virtual processors
 how many 12-23
 NUMAIOVPS parameter 12-23
 performance of 30-19
 when used 12-23

Allocating disk space
 extent 10-13
 for mirrored data 24-5
 initial configuration 3-26
 procedure 11-3
 types of disk space 3-12

ALTER TABLE statement
 mentioned 4-18
 restriction of use 31-15

Alternate dbservername 35-11

ANSI-compliant transaction logging. *See* Logging

Application
 client. *See* Client application.
 developer, role of 1-9

Architecture, dynamic scalable 12-3

Archive
 and backups 1-6
 checkpoint during 14-50
 description of 1-7
 reducing size of 10-19
 reserved page information 40-12
 tools for 3-23
 using ontape 37-70
 with multiple residency 6-7

Archive configuration file 42-5, 42-8

Archive interface attributes file 42-9

Archiving
 strategy 3-4

ARC_DEFAULT environment variable
 mentioned 6-7
 setting with informix.rc 3-31

ARC_KEYPAD environment variable
 42-8, 42-9

Assertion failure
 and data corruption 27-8
 description of 27-3
 determining cause of 27-9
 DUMPCNT parameter 35-17
 DUMPCORE parameter 35-17
 DUMPSHMEM parameter 35-18
 during consistency checking 27-6
 during processing of user request 27-7
 form in message log 27-6

Assertion failure file
 af.xxx 27-6
 gcore.xxx 27-6
 list of 27-6
 shmем.xxx 27-6, 42-8

Asynchronous I/O
 and NUMAIOVPS parameter 35-30
 write requests for mirrored data 23-8

Attaching to shared memory
 additional segments 14-13
 client to communications portion 14-11
 description of 14-10
 first segment 14-12
 OnLine utilities 14-11
 virtual processors 14-11
 virtual processors and key value 14-12

Audit file
 directory 35-7
 size 35-7

Audit mode 12-31

Audit records
 and sysaudit table 36-9
 error 35-6

Auditing 1-8, 35-7, 36-8, 36-9

Automatic recovery, by two-phase
 commit protocol 32-9

Availability
 and critical media 10-15
 as goal in efficient disk layout 10-30
 sample disk layout 10-37

B

Backup
 displaying contents of 37-23
 during practice sessions 3-17
 freeing a log file 19-11

-
- hot site 25-4
 - in a production environment 3-14
 - mentioned 10-29
 - of blobs 10-19
 - planning for 3-4
 - restoring with ontape 37-75
 - starting continuous 37-74
 - backup
 - strategy 3-4
 - Backups
 - changes to the database 1-6
 - of transaction records 1-6
 - with multiple residency 6-7
 - Bad-sector mapping, absence of 1-10
 - Before-image
 - contents 22-5
 - described 20-3
 - flushing of 14-39
 - in physical log buffer 14-39
 - Before-image journal
 - See* Physical log. 17
 - Big buffers
 - and regular buffers 14-27
 - description of 14-27
 - Big-remainder page 40-36
 - Binding CPU virtual processors
 - and AFF_NPROCS parameter 35-8
 - benefit of 12-9
 - parameters 12-19
 - Bit-map page
 - 2-bit values describing fullness 40-23
 - 4-bit values describing fullness 40-23
 - component of a tbspace 10-25
 - component of an extent 40-25
 - component of the tbspace tbspace 40-19, 40-20
 - description of 40-22
 - location in extent 40-22
 - of a blob space 40-59
 - role of 2-bit pages 40-22
 - role of 4-bit page 40-23
 - structure of 40-22
 - types of entries 40-22
 - Blob
 - absence of scanning or compression 1-11
 - and physical logging 20-4
 - creating in a blob space 14-53
 - data types 1-8
 - effect of Committed Read isolation 14-51
 - effect of Dirty Read isolation 14-51
 - entering 1-11
 - how stored 40-55
 - how updated 40-56
 - illustration of blob space storage 14-54
 - in physical and logical log 10-19
 - modified by creating new blob 40-56
 - monitoring for fullness 10-19
 - monitoring in a blob space 29-53
 - monitoring in a db space 29-56
 - page structure 40-57
 - role of blob descriptor in modifications 40-56
 - role of blob descriptor in storage 40-56
 - role of blob timestamps 14-50
 - size limitations 40-57
 - storage efficiency of 40-55
 - storage on disk 40-56
 - when created 14-53, 40-56
 - when modified 40-56
 - writing to a blob space 14-52
 - Blob descriptor
 - associated with blob timestamp 14-50
 - created during blob storage 14-53
 - description of 40-33, 40-56
 - pointer becoming obsolete 14-51
 - structure and function 40-56
 - Blobpage
 - average fullness statistics 37-12
 - blobpage size 40-54
 - blobpage size and storage efficiency 11-17
 - components of page header 40-61
 - description of 10-10
 - determining fullness 11-17
 - freeing deleted pages 18-16
 - fullness terminology explained 11-19
 - interpreting average fullness 11-19
 - oncheck -pB display explained 11-18
 - relationship to chunk 10-10
 - size of 10-10
 - sizing for performance 11-16
 - sizing recommendations 10-11
 - specifying size of 40-54
 - storage statistics 11-17
 - structure and storage 40-54, 40-60
 - writes bypass shared memory 14-52
 - Blobspace
 - activating 18-15

- adding a chunk 11-10, 11-14
- bit-map page 40-59
- blob buffers 14-53
- blob storage 40-56
- blob timestamps 14-50
- blobpage structure 40-60
- creating 3-28, 11-12
- creating using ON-Monitor 11-13
- creating using onspaces 11-14
- description of 10-19
- dropping using ON-Monitor 11-16
- dropping using onspaces 11-16
- dropping, initial tasks 11-15
- free-map page
 - description of 40-59
 - location in blobspace 40-55
 - role in blobpage logging 18-17, 40-59
 - role in tracking blobpages 14-54
 - tracked by bit-map 40-59
- illustration of writing a blob 14-53
- logical-log administration tasks for 18-15
- management of 10-20
- migrating to INFORMIX-SE 31-14
- mirroring 10-19
- moving, with onunload/onload 31-8
- multiple residency 6-6
- overhead pages 18-16
- page types 40-59
- purpose of 10-19
- restriction concerning dropping 37-42
- storage efficiency 11-17
- storage statistics 11-17
- structure 40-54
- structure of blobspace mirror chunk 40-15
- writing data to 14-52

- blobspace 11-17
- Block device 10-6
- Boot file. *See* Startup script
- Btree cleaner list 40-52
- Btree indexing 40-43
- Buffer
 - access-level flag bits 37-51
 - big buffers 14-27
 - blobpage buffer 14-53
 - concurrent access 14-36
 - current lock-access level for 14-19
 - data replication 14-25, 25-9
 - dirty 14-39
 - exclusive mode 14-31
 - flushing 14-39
 - how a thread accesses a buffer page 14-36
 - how a user thread acquires 14-33
 - lock types 14-31
 - lock-access level of 14-37
 - logical-log buffer 14-24
 - minimum requirement 14-19
 - monitoring statistics and use of 29-15
 - not dirty 14-39
 - page-type codes 37-50
 - physical-log buffer 14-25
 - reading from disk 14-37
 - regular buffer 14-23
 - releasing if no thread waiting 14-38
 - releasing if thread waiting 14-38
 - releasing when modified 14-38
 - releasing when not modified 14-37
 - share lock 14-31
 - status 14-18
 - synchronizing flushing 14-39
 - threads waiting for 14-19
 - tuning for performance 30-11
 - write types during flushing 14-42
- Buffer flushing
 - description of 14-39
 - how synchronized 14-41
- Buffer pool
 - bypassed by blobspace data 14-52
 - cleaning efficiency affected by checkpoints 30-14, 30-16
 - contents of 14-23
 - data-replication buffer 14-23
 - description of 14-23
 - flushing 14-39
 - LRU queues management 14-33
 - monitoring activity 29-19
 - read-ahead 14-35
 - synchronizing buffer flushing 14-41
- Buffer table
 - contents of 14-18
 - description of 14-18
 - LRU queues 14-32
- Buffered transaction logging. *See* Logging
- BUFFERS parameter
 - description of 14-24, 35-9
 - mentioned 3-24
 - tuning for performance 30-12

-
- BUFFSIZE. *See* Page size
 - buildsmi
 - script 36-4
 - BYTE data type
 - Committed Read isolation 14-52
 - Dirty Read isolation 14-51
 - migrating from INFORMIX-SE 31-16
 - migrating to INFORMIX-SE 31-14
 - requires 4-bit bit map 40-23, 40-25
 - storage on disk 10-16
 - Byte lock 14-21
 - B+ tree structure 40-43

 - C**
 - Cache. *See* Shared-memory buffer pool
 - Caching percentages
 - description of 30-4
 - mentioned 30-12, 30-14, 30-15
 - Cascading deletes 16-4
 - Cataloger, ON-Archive, mentioned 6-7
 - Cautions 28-3
 - CHAR data type, changing locale 31-17
 - Character-special interface 10-6
 - Checkpoint
 - and chunk writes 14-43
 - and flushing of regular buffers 14-39
 - and logical-log file 14-47
 - and physical-log buffer 14-47, 20-9
 - description of 14-47
 - during an archive 14-50
 - events that initiate 14-47
 - force with onmode -c 37-30
 - forcing, to free logical-log file 15-16
 - frequency trade-offs 30-16
 - main events during 14-48
 - mentioned 9-8
 - monitoring activity 29-11
 - page-cleaning parameters affect
 - frequency 30-14, 30-16
 - reserved page information 40-8
 - role in data replication 25-13
 - role in fast recovery 14-49, 22-5, 22-6
 - step in shared-memory initialization 9-8
 - updating the reserved pages 40-6
 - Chunk
 - activity during mirror recovery 23-7
 - adding to a blobospace 11-14
 - adding to a dbspace 11-10
 - adding to mirrored dbspace 24-8
 - adding using ON-Monitor 11-11
 - adding using onspaces 11-11
 - changing mirror chunk status 24-8
 - checking for overlap 37-11
 - chunk status as PD 27-8
 - cooked versus raw 10-5
 - creating a link to the pathname 11-6, 24-5
 - defining multiple in a partition 11-5
 - definition of 3-12
 - description of 10-4
 - disk layout guidelines 10-31
 - dropping from a blobospace 11-15
 - dropping from a dbspace 11-14
 - exceeding size limits with LVM 10-44
 - extent interleaving 11-20
 - free-list page 40-13, 40-14, 40-16, 40-17
 - initial chunk of dbspace 40-4
 - initial mirror offset 35-26
 - I/O errors during processing 27-8
 - management of 11-19
 - maximum number of 10-5, 35-10
 - maximum size of 10-5
 - mirror chunk reserved page information 40-11
 - monitoring 11-19, 29-43, 29-46, 36-9, 36-10
 - multiple residency 6-6
 - name, when allocated as raw device 10-6
 - pathname stored 40-10
 - purpose of 10-5
 - recovering a down chunk 24-8
 - relation to extent 10-13
 - relinking after disk failure 24-10
 - reserved page information 40-10
 - structure
 - additional dbspace chunk 40-14
 - initial dbspace chunk 40-13
 - mirror chunk 40-15
 - using a symbolic link for the pathname 35-36
 - Chunk table
 - and mirroring 14-19
 - description of 14-19
 - maximum entries 14-19
 - Chunk write
 - checkpoints 14-43
 - monitoring 29-20

-
- CHUNKS parameter
 - description of 35-10
 - mentioned 3-24
 - relationship to DBSPACES 35-13
 - CKPTINTVL parameter
 - description of 35-10
 - initiating checkpoints 14-47
 - mentioned 3-25
 - tuning for performance 30-16
 - Classes of virtual processor 12-5
 - CLEANERS parameter
 - description of 35-11
 - purpose of 14-21
 - tuning for performance 30-14
 - Client
 - description of 1-4
 - remote 4-9
 - Client application
 - connecting to OnLine 4-11
 - redirecting in data replication 25-19
 - testing 5-4
 - Client/server architecture
 - client/server connection 1-4
 - description of 1-4
 - Client/server configuration
 - connections 4-3
 - example
 - 6.0 client with 5.0 server 4-30
 - 6.0 relay module 4-26
 - INFORMIX-STAR, example 4-29
 - local loopback 4-21
 - multiple connection types 4-23
 - multiple OnLine servers 4-25
 - multiple residency 4-25
 - network connection 4-21
 - shared memory 4-20
 - using IPX/SPX 4-23
 - listen and poll threads 12-26
 - local loopback 4-6
 - remote host 4-5
 - shared memory 4-4
 - types of 4-3
 - Code examples, conventions Intro-8, 8
 - Cold restore
 - number of recovery threads 35-31
 - Committed Read isolation level
 - data-consistency checks 14-51
 - role of blob timestamps 14-51
 - Communication configuration file. *See* ONCONFIG configuration file
 - Communications portion (shared memory)
 - contents of 14-29
 - description of 14-29
 - how client attaches 14-10
 - size of 14-30
 - Communication, client to database server. *See* Connectivity
 - Compactness, of index page 35-19
 - Concurrency control 14-30
 - Condition segment 31-15
 - Configuration
 - learning environment 3-10
 - monitoring 29-9
 - overview 3-8
 - parameter overview 3-19
 - planning for OnLine 3-4
 - production environment 3-17
 - template file 3-8
 - using 6.0 relay module 4-26
 - Configuration file
 - and multiple residency 5-3, 6-4
 - archive 42-5
 - connectivity 4-7
 - description of 3-13
 - onconfig.std 3-8
 - reserved page information 40-8
 - warning about multiple residency 6-4
 - Configuration parameter
 - ADTERR 35-6
 - ADTMODE 35-7
 - ADTPATH 35-7
 - ADTSIZE 35-7
 - AFF_NPROCS 35-8
 - AFF_SPROC 35-8
 - and initial chunk of root dbspace 10-17
 - BUFFERS 35-9
 - CHUNKS 35-10
 - CKPTINTVL 35-10
 - CLEANERS 35-11
 - CONSOLE 3-19, 35-11
 - DBSERVERALIASES 4-18, 35-11
 - DBSERVERNAME 3-15, 3-19, 4-12, 4-17, 35-12
 - DBSPACES 35-13
 - DBSPACETEMP 35-13
 - DEADLOCK_TIMEOUT 35-14

displayed
 in data-replication screen 34-15
 in diagnostics screen 34-16
 in initialization screen 34-12
 in shared-memory screen 34-13
 in virtual processor screen 34-14
 DRAUTO 35-15
 DRINTERVAL 35-15
 DRLOSTFOUND 35-16
 DRTIMEOUT 35-16
 DUMPCNT 35-17
 DUMPCORE 35-17
 DUMPDIR 35-17
 DUMPGCORE 35-18
 DUMPSHMEM 35-18
 FILLFACTOR 35-19
 for archiving 3-23
 for diagnostic information 27-9
 for multiple residency 5-4
 for ontape utility 3-23
 for physical logging 3-22
 identification 3-20
 LOCKS 35-19
 LOGBUFF 35-20
 LOGFILES 35-20
 LOGSIZE 35-21
 LOGSMAX 35-22
 LRUS 35-22
 LRU_MAX_DIRTY 35-22
 LRU_MIN_DIRTY 35-23
 LTAPEDEV 3-14, 3-19, 35-24
 LTAPESIZE 35-24
 LTXEHWM 35-25
 LTXHWM 35-25
 MIRROR 35-26
 MIRROROFFSET 35-26
 MIRRORPATH 35-27
 MSGPATH 3-15, 3-19, 35-27
 MULTIPROCESSOR 35-27
 NETTYPE 4-17, 35-28
 NOAGE 35-30
 NUMAIOVPS 35-30
 NUMCPUVPS 35-31
 OFF_RECVRY_THREADS 35-31
 ON_RECVRY_THREADS 35-32
 PHYSBUFF 35-32
 PHYSDBS 35-33
 PHYSFILE 35-33
 RA_PAGES 35-34
 RA_THRESHOLD 35-34
 RESIDENT 35-35
 ROOTNAME 3-20
 ROOTOFFSET 35-36
 ROOTPATH 3-14, 3-19, 35-35, 35-36
 ROOTSIZE 3-20, 35-36
 SERVERNUM 3-15, 3-19, 14-11, 35-37
 shared memory 3-24, 15-3
 SHMBASE 14-11, 35-38
 SHMVIRTSIZE 35-39
 SINGLE_CPU_VP 35-40
 STACKSIZE 35-41
 STAGEBLOB 35-41
 TAPEBLK 35-41
 TAPEDEV 3-14, 3-19, 35-42
 TAPESIZE 35-44
 TBLSPACES 35-44
 TRANSACTIONS 35-45
 TXTIMEOUT 35-45
 USEOSTIME 30-20, 35-46
 config.arc file
 description of 42-5
 mentioned 3-23
 with multiple residency 6-7
 CONNECT statement
 example 4-18
 mentioned 4-11, 4-18
 USER clause 4-9
 Connecting
 description of 12-26
 example with CONNECT 4-11
 methods 12-24
 to non-Informix databases 1-8
 with sockets 12-24
 with TLI 12-24
 Connection
 5.0 relay module 4-29
 6.0 client with 5.0 server, example 4-30
 client/server, types of 4-3
 from 5.0 server to 6.0 server 4-27
 IPX/SPX 4-23
 local loopback, definition of 4-6
 local loopback, example 4-21
 multiple connection types
 example 4-23
 multiple residency, example 4-25
 network, description of 4-5
 network, example 4-21
 network, when to use 4-5
 security restrictions 4-9
 shared memory, description of 4-4
 TCP/IP 4-8
 to client application 4-3
 to database server 4-3

- Connectivity
 - configuration file 4-7
 - configuration parameters 4-17
 - file, sqlhosts 3-8
- Consistency checking
 - corruption of data 27-7
 - data and overhead 27-4
 - index corruption 27-8
 - monitoring for data inconsistency 27-6
 - overview 27-3
 - periodic tasks 27-3
 - validating extents 27-5
 - validating indexes 27-5
 - validating reserved pages 27-4
 - when to schedule 27-4
- Console messages 29-7
- CONSOLE parameter
 - changing 29-7
 - description of 35-11
 - in a production environment 3-19
 - mentioned 3-23
- Constraint Name segment 31-15
- Constraint, deferred checking 16-4
- Contention. *See* Disk contention
- Context switching, description of 12-10
- Control structures
 - description of 12-10
 - queues 12-14
 - session control block 12-10
 - stacks 12-12
 - thread control block 12-10
- Conventions
 - boldface font Intro-5, 5
 - command-line syntax Intro-5, 5
 - computer font Intro-5, 5
 - example code Intro-8, 8
 - italics font Intro-5, 5
 - keywords Intro-5, 5
 - railroad diagrams Intro-5, 5
- Conversion
 - between locales 31-18
 - during initialization 9-7
- Cooked file space
 - and buffering 10-7
 - compared with raw space 10-6
 - contiguity of space 10-7
 - description of 3-12, 10-5, 10-6
 - directory for 3-12
 - for static data 10-7
 - how to allocate 11-4
 - in data storage 10-3
 - OnLine management of 10-6
 - rationale for using 10-7
 - reliability 10-8
 - steps to prepare 3-13
 - warning 10-6
- Coordinating database server
 - and automatic recovery 32-11
 - description of 32-5
- Coordinator recovery mechanism 32-11
- Core dump 27-10
 - and DUMPCORE configuration parameter 35-17
 - contained in core file 42-5
 - contents of gcore.xxx 27-6
 - when useful 27-10
 - See also* DUMPCNT, DUMPPDIR, DUMPGCORE, DUMPSHMEM
- core.pid.cnt file 35-18
- Corruption
 - corrective actions 27-8
 - determining if exists 27-8
 - I/O errors from a chunk 27-8
 - symptoms of 27-8
- CPU time tabulated 37-63
- CPU virtual processor
 - adding and dropping in on-line mode 12-18
 - AFF_NPROCS parameter 12-18
 - AFF_SPROC parameter 12-18
 - and poll threads 12-25, 12-26
 - and SINGLE_CPU_VP parameter 35-40
 - binding 35-8
 - description of 12-16
 - how many 12-16
 - more than two CPUs 12-17
 - on a multiprocessor machine 12-17
 - on a single processor machine 12-17
 - performance of 30-18
 - preventing priority aging 12-18
 - the NUMCPUVPS parameter 12-16
 - types of threads run by 12-16
- CREATE DATABASE statement
 - effect of locale 31-17
 - mentioned 31-15
- CREATE INDEX statement
 - mentioned 31-15

- using FILLFACTOR 35-19
- CREATE SCHEMA statement 31-15
- CREATE TABLE statement
 - mentioned 4-18
 - migrating to INFORMIX-SE 31-14
 - use with dbload 31-9
- Critical media
 - and archiving 10-36
 - mirroring 10-40
 - storage of 10-15
- Critical section of code
 - and checkpoints 14-48
 - description of 14-46
 - how to determine 37-31
 - related to size of logical log 18-6
 - related to size of physical log 20-5
- cron jobs, warning about 3-29, 42-9

D

- Data block
 - See Page.* 16
- Data consistency
 - fast recovery 22-3
 - monitoring for 27-6
 - symptoms of corruption 27-8
 - timestamps 14-50
 - verifying consistency 27-4
- Data definition statements, when logged 16-6
- Data files. *See* Logging.
- Data management 10-6
- Data manipulation statements, when logged 16-6
- Data replication
 - actions to take if primary fails 25-17
 - actions to take if secondary fails 25-16
 - administration of 26-12
 - advantages of 25-4
 - and the PAGE_ARCH reserved page 40-12
 - automatic switchover 25-17
 - changing database server mode 26-15
 - changing database server type 26-16
 - client redirection
 - comparison of different methods 25-28
 - handling within an application 25-26
 - using DBPATH 25-20

- using INFORMIXSERVER 25-25
- using sqlhosts file 25-22
- configuring connectivity for 26-8
- database and data requirements for 26-5
- database server configuration
 - requirements for 26-5
- description of 1-7, 25-3
- designing clients to use secondary database server 25-29
- detecting failures of 25-15
- DRAUTO parameter 25-17, 35-15
- DRINTERVAL parameter 25-10, 25-11
- DRLOSTFOUND parameter 25-11
- DRTIMEOUT parameter 25-15
- flush interval 35-15
- hardware and operating system
 - requirements for 26-4
- how it works 25-8
- how updates are replicated 25-9
- importance of reliable network 25-18
- information in sysdri table 36-13
- initial replication 25-8
- lost and found file 35-16
- lost and found transactions 25-11
- manual switchover 25-19
- mentioned 16-5
- monitoring status 29-58
- planning for 26-3
- possible failures 25-14
- read only mode 7-4
- restarting after failure 25-18, 25-19, 26-20
- restoring system after media failure 26-18
- role of checkpoint 25-13
- role of log records 25-9
- role of primary database server 25-4
- role of secondary database server 25-4
- role of temporary dbspaces 25-31
- setting up 26-4
- specialized threads 25-13
- starting 26-9
- synchronization 25-14
- wait time for response 35-16
- with asynchronous updating 25-11
- with synchronous updating 25-10

- Data row
 - and rowid 40-34
 - big-remainder page 40-36
 - blob descriptor component 40-56
 - forward pointer 40-34, 40-36

-
- home page 40-33, 40-36
 - how OnLine locates 40-34
 - storage strategies 40-33
 - storing data on a page 40-35
- Data storage
- control of 10-15
 - overview 10-3
 - types of 10-3
 - See also* Disk space.
- Data Type segment 31-15
- See also* Disk space.
- Database
- controlling storage location 10-22
 - creating, what happens on disk 40-63
 - description of 10-20
 - effects of creation 40-63
 - estimating size of 10-30
 - information in sysdatabases table 36-11
 - limits 10-21
 - location of 10-20
 - migration *See* Migration
 - monitoring 29-36
 - moving, using onunload/onload 31-7
 - owner, in sysmaster database 36-11
 - purpose of 10-20
 - recovery *See* Recovery
 - revert to 5.0 format 37-35
 - size limits 10-21
 - stores6 demonstration database Intro-9, 9
 - tuning. *See* Performance tuning
- Database administrator
- role of 1-10
- Database format, change with onmode 37-35
- Database logging status
- ANSI-compliant, description 16-9
 - buffered, description 16-9
 - changes permitted 17-3
 - changes, general info 17-4
 - changing buffering status
 - using ON-Archive 17-6
 - using ontape 17-7, 37-73
 - ending logging
 - using ON-Archive 17-6
 - using ontape 17-7
 - in a distributed environment 16-10
 - making ANSI-compliant
 - using ON-Archive 17-6
 - using ontape 17-8
 - modifying
 - using ON-Archive 17-5
 - using ON-Monitor 17-8
 - using ontape 17-6
 - set after moving data 31-14
 - setting 16-7
 - turning on logging
 - using ON-Archive 17-5
 - using ontape 17-7
 - unbuffered, description 16-8
 - who can change 16-10
- Database Name segment 31-15
- Database object, ways to create 31-9
- Database schema. *See* Schema file
- Database server
- connection to 4-3
 - description of 1-3
 - remote 4-10, 36-19
- Database server name. *See* dbserveraname
- DATABASE statement, mentioned 4-18
- Database tblspace
- entries 40-21
 - location in root dbspace 40-4, 40-20
 - relation to systable 40-63
 - structure and function 40-20
 - tblspace number 40-21
- Data-recovery mechanisms
- fast recovery 22-3
 - mirroring 23-3
- Data-replication buffer 14-23, 14-25, 25-9
- DB-Access
- create a database object 31-9
 - testing configuration 4-30
 - to test OnLine features 3-17
- dbexport utility
- mentioned 31-4
 - privileges required 31-13
 - schema file 31-13
 - steps for using 31-13
- dbimport utility
- mentioned 31-4
 - moving data 31-13
 - steps for using 31-13
- dbload utility
- bad-record treatment 31-12
 - external data 31-12
 - list of features 31-11
 - mentioned 31-4

- options 31-11
- steps for using 31-12
- DBNETTYPE environment variable 4-28
- DBPATH environment variable
 - moving to SE 31-14
 - use in automatic redirection 25-20
- dbschema utility
 - description 31-10
 - mentioned 31-4
- DBSERVERALIASES parameter
 - description of 4-18, 35-11
 - example 4-18
 - in sqlhosts file 4-12
 - mentioned 3-20
 - multiple connection types example 4-23
 - specifying network protocols 12-25
- dbservername
 - choosing 3-11
 - conflict with INFORMIX-SE 35-12
 - description of 35-12
 - field in sqlhosts file 4-12
 - in CONNECT statement 4-11
 - mentioned 1-4
 - purpose of 4-12
- DBSERVERNAME parameter
 - associated protocol 12-25
 - description of 4-17, 35-12
 - in a learning environment 3-15
 - in a production environment 3-19
 - in sqlhosts file 4-12
 - mentioned 3-20
 - multiple residency 6-5
 - specifying network protocols 12-25
 - virtual processor for poll thread 12-25
- Dbospace
 - adding a chunk to 11-10
 - adding a mirrored chunk 24-8
 - as link between logical and physical units of storage 10-15
 - bit-map page 40-22
 - blob page structure 40-57
 - blob storage 40-56
 - blob timestamps 14-50
 - creating a temporary 11-8
 - creating during initial configuration 3-28
 - creating with ON-Monitor 11-9
 - creating with onspaces 11-9
 - creating, overview 11-8
 - description of 10-15
 - dropping a chunk from 11-14
 - dropping using ON-Monitor 11-16
 - dropping using onspaces 11-16
 - dropping, overview 11-15
 - identifying the dbspace for a table 40-19
 - initial dbspace 10-17
 - list of structures contained in 40-13
 - maximum number of 35-13
 - mirror chunk information 40-7
 - mirroring if logical log files included 23-6
 - monitoring blobs 29-56
 - monitoring with SMI 36-12
 - multiple residency 6-6
 - page header 40-30
 - primary chunk information 40-7
 - purpose of 10-15
 - reserved page information 40-9
 - root dbspace defined 10-17
 - root name 35-35
 - shared-memory table 14-20
 - starting to mirror 24-6
 - storage 40-4
 - structure 40-4, 40-13, 40-14
 - structure of additional dbspace chunk 40-14
 - structure of chunk free-list page 40-16
 - structure of mirror chunk 40-15
 - structure of nonroot dbspace 40-13
 - structure of tblspace tblspace 40-17
 - temporary 10-18
 - usage report 40-4
- DBSPACES parameter
 - description of 35-13
 - mentioned 3-24
- DBSPACETEMP environment variable 10-25
- DBSPACETEMP parameter
 - and load balancing 10-34
 - description of 35-13
 - if not set 10-25
 - mentioned 10-18
 - relationship to DBSPACETEMP environment variable 10-24
- Deadlock. *See* DEADLOCK_TIMEOUT parameter.
- DEADLOCK_TIMEOUT parameter
 - description of 35-14
 - in two-phase commit 32-34

- mentioned 3-25
- Default configuration file 3-9, 9-6, 42-7
- Deferred checking of constraints 16-4
- DEFINE SPL statement 31-15
- Delete flag
 - described 40-44
 - possible values 40-44
- Demonstration database
 - copying Intro-10, 10
 - installation script Intro-9, 9
 - overview Intro-9, 9
- Device 10-6
- Diagnostic information
 - and disk space restraints 27-10
 - collecting 27-9
 - parameters to set 27-9
- Diagnostic messages. *See* Message log
- Dictionary cache 14-28
- Dirty buffer
 - description of 14-39
- Dirty Read isolation level
 - data-consistency checks 14-51
 - role of blob timestamps 14-51
- Disk contention
 - and high-use tables 10-31
 - and multiple disk devices 10-33
 - of critical media 10-35
 - reducing 10-30
- Disk failure 1-7
- Disk I/O
 - kernel asynchronous I/O 12-20
 - logical log 12-20
 - managing 10-3
 - operating system I/O 10-6
 - physical log 12-20
 - priorities 12-21
 - raw I/O 10-6
 - reads from mirrored chunks 23-8
 - virtual processor classes 12-20
 - writes to mirrored chunks 23-8
- Disk layout
 - and archiving 10-32, 10-36
 - and logical volume managers 10-44
 - and mirroring 10-31
 - and table isolation 10-31
 - for optimum performance **10-31**, 30-6
 - sample disk layouts 10-37
 - trade-offs 10-37
- Disk page
 - before-images in physical log 14-39
 - function of timestamp pairs 14-50
 - logical page number 40-34
 - number to read ahead 35-34
 - page compression 40-32, **40-41**
 - physical page number 40-34
 - read ahead 14-35
 - storing data on a page 40-35
 - structure
 - blob space blobpage 40-25
 - dbspace page 40-30
 - types of pages in an extent 40-25
- Disk space
 - allocating
 - cooked file space 11-4
 - raw disk space 10-7, 11-5
 - when a database is created 40-63
 - when a table is created 40-64
 - allocation for system catalogs 40-63
 - chunk free-list page 40-16
 - creating a link to chunk pathname
 - 11-6
 - described 11-3
 - efficient storage of blobs 11-17
 - estimating size of 10-27
 - extent interleaving 11-20
 - guidelines for layout 10-31
 - initialization 9-7
 - definition of 9-3, 11-7
 - with new database server 3-27
 - with oninit 11-7, 37-16
 - with ON-Monitor 11-7
 - layout guidelines 10-30
 - list of structures 40-3
 - multiple residency 6-6
 - offsets for chunk pathnames 11-5
 - optimal storage of tables 10-33
 - optimizing temporary space layout
 - 10-34
 - page compression 40-32, **40-41**
 - raw devices versus cooked files 10-5
 - reclaiming space 11-21
 - requirements
 - for production environment 10-30
 - for root dbspace 10-27
 - strategies for improving performance
 - 10-30
 - tracking
 - available space in a blob space
 - 40-59
 - available space in a chunk 40-16

- free pages with bit-map page 40-22
- usage by tblspace 10-25
- Disk volume inconsistency information 42-9
- Distributed databases 1-8
- Distributed queries, description of 1-8
- Distributed transaction
 - and two-phase commit protocol 32-3
 - determining if inconsistently implemented 33-5
 - mentioned 16-4
- Documentation
 - notes Intro-8, 8
 - other useful Intro-4, 4
- DRAUTO parameter
 - description of 35-15
 - role in recovering from data-replication failure 25-17
- DRINTERVAL parameter
 - description of 35-15
 - setting for asynchronous updating 25-11
 - setting for synchronous updating 25-10
- DRLOSTFOUND parameter
 - description of 35-16
 - use with data replication 25-11
- Dropping CPU virtual processors 12-9
- DRTIMEOUT parameter
 - description of 35-16
 - role in detecting data replication failures 25-15
- dr.lostfound file 35-15
- DUMPCNT parameter 35-17
- DUMPCORE parameter 35-17
- DUMPDIR parameter
 - af.xxx assertion failure file 42-4
 - and consistency checking 27-6
 - and shmем file 42-8
 - description of 35-17
 - gcore file 42-5
- DUMPGCORE parameter 27-6, 35-18
- DUMPSHMEM parameter 27-6, 35-18
- Dynamic scalable architecture
 - advantages 12-3
 - description of 12-3

E

- Environment configuration file 42-5
- Environment variable
 - ARC_DEFAULT 3-31
 - ARC_KEYPAD 42-8, 42-9
 - DBNETTYPE 4-28
 - for NLS 3-18
 - for users of client applications 3-30
 - in a learning environment 3-11
 - in a production environment 3-18
 - INFORMIXDIR 3-9, 3-28, 4-27
 - INFORMIXSERVER 3-9
 - INFORMIXSHMBASE 14-11
 - INFORMIXTERM 3-9
 - ONCONFIG 3-9
 - overview 3-9
 - PATH 3-9, 3-28
 - PSORT_DBTEMP 30-10
 - PSORT_NPROCS 30-9
 - required by application 3-9
 - SQLEXEC 4-27
 - SQLRM 4-27
 - SQLRMDIR 4-27
 - TBCONFIG 42-7
 - TERM 3-9
 - TERMCAP 3-9
- Error messages
 - documentation for Intro-4, 4
 - for two-phase commit protocol 32-27
 - I/O errors on a chunk 27-8
 - /etc/hosts file 4-8
 - /etc/services file 4-8
- Example
 - 6.0 relay module 4-26
 - connecting 6.0 client to 5.0 server 4-31
 - DBSERVERALIASES and sqlhosts file 4-18
 - DBSERVERALIASES parameter 4-18
 - how page cleaning begins 14-34
 - IPX/SPX connection 4-23
 - local loopback connection 4-21
 - multiple connection types 4-23
 - NETTYPE parameters for tuning 35-29
 - relay module with three servers 4-28
 - shared-memory connection 4-20
 - SQLEXEC environment variable 4-27
 - sqlhosts file 4-11
 - TCP/IP connection 4-22
 - /etc/services file entry 4-8

Exclusive lock (buffer), description of 14-31

exit codes
 ontape utility 37-71

Explicit temporary table 10-24

Expression segment 31-15

Extent
 automatic doubling of size 40-27
 default size 40-24
 description of 10-11
 disk page types 40-25
 how OnLine allocates 10-13
 information in sysextents table 36-13
 initial size 10-12, 40-24
 interleaving 10-26, 11-20
 key concepts concerning 10-13
 merging 40-28
 monitoring 29-50
 next extent allocation 40-27
 next extent allocation strategies 40-29
 next extent size 10-12, 40-24, 40-27
 optimizing table 10-34
 procedure for allocating 40-27
 purpose of 10-11
 reclaiming space 11-21
 relationship to chunk 10-13
 size limitations 40-24
 size, migrating to INFORMIX-SE 31-14
 structure 10-12, 40-24
 tracking free pages using bit-map page 40-22
 validating consistency 27-5

F

Fast recovery
 description of 1-7, 22-3
 details of process 22-5
 effects of buffered logging 22-4
 how OnLine detects need for 22-4
 mentioned 7-4, 9-8, 16-4
 no logging 22-4
 purpose of 22-3
 role of checkpoint 14-49
 role of PAGE_CHK reserved page 22-6
 when initiated 22-4
 when needed 22-3
 when occurs 22-3

Fault tolerance
 archives and backups 1-6
 data replication 1-7, 25-4, 25-6
 fast recovery 1-7, 22-3
 mirroring 1-7

File
 archive configuration file 42-5
 communication configuration 4-7
 configuration 3-8
 config.arc 6-7, 42-5
 connectivity configuration 4-7
 core.pid.cnt 35-18
 default configuration file 42-7
 dr.lostfound 35-15
 gcore 35-17
 informix.rc environment file 42-5
 network configuration 4-7
 network security 4-9
 oncatlgr.out.pidnum 42-6
 oncfg_servername.servernum 9-8, 42-8
 onconfig, during initialization 9-5, 9-6
 onconfig.std
 description of 3-8, 42-7
 during initialization 9-6
 sample file 42-10
 oper_deflt.arc 6-7, 42-8
 private environment file 42-5
 shmем.pid.cnt 35-19
 shmем.xxx 42-8
 sqlhosts 3-8, 4-10
 status_vset_volnum.itgr 42-9
 summary of OnLine files 42-3
 sysfail.pidnum 42-9
 tctermcap archive attributes file 42-9
 template for configuration file 42-7
 VP.servername.xxC 42-9
 .informix 42-5
 .infos.dbservername 42-6
 .inf.servicename 42-6
 .netrc 4-9
 .rhosts 4-9
 /etc/hosts.equiv 4-9
 /etc/shadow 4-9

File I/O. *See* Disk I/O

FILLFACTOR parameter
 controlling how indexes filled 40-53
 description of 35-19

-
- Fixed format data, moving with dbload 31-12
 - FLRU queues
 - and reading a page from disk 14-37
 - and releasing buffer 14-38
 - description of 14-32
 - See also* LRU queues
 - Flushing
 - buffers 14-39
 - data-replication buffer, maximum interval 35-15
 - of before-images 14-39
 - Forced residency
 - initialization 9-9
 - start/end with onmode 37-30
 - Forcing a checkpoint. *See* Checkpoint
 - Foreground write
 - description of 14-43
 - monitoring 29-20
 - Format of database, change with onmode 37-35
 - Forward pointer
 - description of 40-34
 - how it can become invalid 14-51
 - role in a blobspace blob storage 40-60
 - role in data storage 40-36
 - role in dbspace blob storage 40-56
 - unaffected by page compression 40-32
 - Free list. *See* Chunk free list
 - Free-map page
 - description of blobspace free-map page 40-59
 - role in blobspace logging 18-17
- G**
- gcore
 - file 27-6
 - utility 35-17, 35-18
 - Global pool, description of 14-29
 - Global transaction identification number 33-6, 33-7
- H**
- Hash table
 - to buffer table 14-19
 - to lock table 14-21
 - to tblspace table 14-22
 - Heaps 14-28
 - Heuristic decision
 - result from independent action 32-19
 - types of 32-20
 - Heuristic end-transaction
 - conditions for 32-25
 - description of 32-25
 - determining effects on transaction 33-4
 - illustration, including logical log records 32-32
 - messages returned by 32-26
 - results of 32-26
 - when necessary 32-25
 - Heuristic rollback
 - conditions resulting in 32-21
 - description of 32-21
 - illustration, including logical log records 32-30
 - indications that rollback occurred 33-4
 - results of 32-22
 - High availability data replication. *See* Data replication
 - Home page 40-33, **40-36**
 - hostname field
 - multiple network interface cards 12-30
 - with IPX/SPX 4-15
 - with shared memory 4-14
 - with TCP/IP communication protocol 4-15
 - Hot site backup. *See* Data replication
- I**
- Identification parameters 3-20
 - Inconsistency information
 - disk volume 42-9
 - how to detect 27-3
 - Index
 - branch node 40-43
 - btrees cleaner list 40-52
 - controlling how filled 40-53
 - delete flag 40-44
 - duplicate key values 40-49
 - effects on structure of item insertion 40-52
 - how created and filled 40-45
 - how deleted items removed 40-52

- insertion of indexed data 40-52
- item described 40-44
- key value locking 40-52
- leaf node 40-43
- repairing structures with oncheck
 - utility 37-6
- reuse of freed pages 40-52
- root node 40-43
- structure of B+ tree 40-43
- validating consistency 27-5
- Index item
 - calculating the length of 40-53
 - defined 40-44
 - example of 40-44
 - how removed 40-52
 - merging 40-53
 - shuffling 40-53
 - when purged 40-44
- Index Name segment 31-15
- Index page
 - compactness 35-19
 - creation of first 40-45
 - effects of creation 40-45
 - effects of inserting item 40-52
 - structure of 40-43
- Infinity item
 - and creation of branch node 40-47
 - defined 40-46
- Informix application development tools
 - Intro-3, 3
- Informix recommendations
 - on allocation of disk space 10-8
 - on consistency checking 27-3
 - on mirroring the physical log 20-7
- INFORMIXDIR environment variable
 - and 5.0 products 4-27
 - in shutdown script 3-29
 - in startup script 3-28
 - mentioned 3-9
 - multiple residency startup script 6-8
- INFORMIX-OnLine Dynamic Server
 - administering 3-5
 - administrator, description of 1-10
 - bad-sector mapping, absence of 1-10
 - blob compression, absence of 1-11
 - blob scanning, absence of 1-11
 - client/server architecture 1-4
 - connecting to 5.0 server from 6.0 server 4-27
 - connecting to multiple servers 4-25, 5-3
 - connection types supported by 4-3
 - description of 1-3
 - distributed queries 1-8
 - fault-tolerant features 1-6
 - features beyond the scope of 1-10
 - high performance of 1-5
 - management of data 10-6
 - message log file 35-27
 - multimedia support 1-8
 - multiple instances 5-3
 - profile statistics 29-13
 - raw-disk management 1-5
 - resident portion of shared memory 35-35
 - security 1-8
 - shut down with UNIX script 3-29
 - testing environment 5-3
 - users 1-9
- INFORMIX-OnLine/Optical multimedia
 - support 1-8
- INFORMIXSERVER environment
 - variable
 - during initialization 3-9
 - multiple versions of OnLine 3-29
 - multiple residency startup script 6-8
 - relation to DBSERVERNAME 35-12
 - use in client redirection 25-25
 - with multiple residency 6-8
- INFORMIXSHMBASE environment
 - variable 14-11
- INFORMIXSTACKSIZE environment
 - variable, purpose of 14-28
- INFORMIXTERM environment variable
 - 3-9
- informix.rc environment file
 - description of 42-5
 - mentioned 42-5
 - multiple residency 6-8
 - use of 3-30
- Initial configuration
 - creating blobspaces and dbspaces 3-28
 - disk layout for production
 - environment 10-30
 - guidelines for root dbspace 10-17
 - raw disk devices versus cooked files 10-5

Initialization

- checkpoint 9-8
- commands 11-7
- configuration changes 9-8
- configuration files 9-5
- control returned to user 9-9
- conversion of internal files 9-7
- disk space 3-27, 9-3, 9-7
- disk space for multiple residency 6-6
- disk structures initialized 40-4
- environment variables to set 3-9
- fast recovery 9-8
- forced residency 9-9
- message log 9-9
- oncfg_servername.servernum file 9-8
- onconfig file 9-6
- onconfig.std file 9-6
- reserved page information 40-7
- shared-memory 9-3
- shared-memory segments 9-6
- SMI tables 9-9
- steps in 9-4
- temporary tablespaces 9-8
- upon completion 9-9
- utilities for 9-4
- virtual processors 9-7

INSERT INTO statement 31-15

Installation

- definition of 3-6
- getting ready 3-7
- upgrading from an earlier version 3-7
- when no other Informix products are present 3-6
- when other Informix products are present 3-6
- when SE is present 3-6

Integrity, data. *See* Consistency checking

Interface attributes for ON-Archive 42-9

Interprocess communication

- in nettype field 4-13
- shared memory for 14-5

Interval, checkpoint 35-10

ipcshm protocol and communications

- portion (shared memory) 14-30

IPC. *See* Interprocess communications

IPX/SPX

- in hostname field 4-15, 4-23
- in nettype field 4-14
- in servicename field 4-16
- multiple residency 6-6

- service, definition of 4-16
- sqlhosts entry 4-23
- support of 1-8
- using 4-9

ISAM calls tabulated 37-62

Isolation level

- Committed Read and blobs 14-51
- Dirty Read and blobs 14-51

I/O errors during processing 27-8

I/O. *See* Disk I/O.

K

KAIO thread 12-22

Kernel asynchronous I/O

- description of 12-22
- non-logging disk I/O 12-20

Key value

- and index items 40-44
- checking ordering with oncheck 37-10
- duplicates 40-49, 40-50
- for shared memory 14-12
- locking 40-52

L

Latch

- description of 14-30
- determining if one is held using onmode 37-31
- displaying information with onstat 37-64
- identifying the resource controlled 37-64
- monitoring statistics and use 29-22
- mutex 12-16, 14-30

Learning environment, configuring 3-10

Level-0 archive

- after moving data 31-7, 31-8, 31-11, 31-14
- after moving NLS data 31-18
- use in consistency checking 27-7

Lightweight processes 12-5

Linking, name of root dbspace 35-36

LIO virtual processors

- description of 12-21
- how many 12-21

List of

- OnLine modes 7-3

- Listen threads
 - and multiple interface cards 12-30
 - description of 12-26
- Load balancing
 - as performance goal 10-30
 - of critical media 10-35
 - through use of DBSPACETEMP 10-34
- LOAD statement, steps for using 31-11
- Local loopback
 - compared with shared-memory connection 4-6
 - connection 4-6, 12-24
 - example 4-21
 - restriction 4-6
- Locale
 - description of 31-17
 - moving data between 31-17
- Lock
 - buffer-access-level flag bits 37-51
 - description of 14-30
 - information in syslocks table 36-13
 - key value locking 40-52
 - maximum time to acquire 35-14
 - migrating to INFORMIX-SE 31-14
 - monitoring 29-23, 37-58
 - type codes 37-58
 - types 14-31
- Lock table
 - contents of 14-20
 - hash table 14-21
 - maximum number of entries 14-21
- Lock-access level
 - of a buffer 14-37
- Locking
 - for multi-processor 35-27
 - when occurs 14-37
 - when released 14-37
- LOCKS parameter
 - description of 35-19
 - mentioned 3-24
 - tuning for performance 30-18
- LOGBUFF parameter
 - and logical log buffers 14-24
 - description of 35-20
 - mentioned 3-21
- LOGFILES parameter
 - changing 19-8
 - description of 35-20
 - mentioned 3-21
- use in logical log size determination 18-7
- Logging
 - activity that is always logged 16-6
 - definition of transaction logging 16-7
 - effect of buffering on logical log fill rate 18-13
 - monitoring activity 29-37
 - OnLine processes requiring 16-4
 - physical logging
 - description of 20-3
 - process of 20-8
 - purpose of 20-3
 - sizing guidelines for 20-5
 - suppression in temporary dbspaces 10-19
 - process for blobSPACE data 18-20
 - process for dbSPACE data 18-18
 - role in data replication 25-9
 - role of blobSPACE free-map page 18-20, 40-59
 - suppression for implicit tables 10-18
 - when to buffer transaction logging 16-10
 - when to use transaction logging 16-9
 - with blob data 16-7
 - See also* Database logging status
- Logical consistency, description of 22-5
- Logical log
 - administration tasks for blobSPACES 18-15
 - checking consistency 37-11
 - configuration parameters 3-21, 19-8
 - description of 14-24, 18-3
 - determining disk space allocated 18-7
 - in rootdbSPACE 40-4
 - maximum number of files 35-22
 - monitoring with SMI 36-14
 - optimal storage of 10-35, 10-36
 - purpose of 1-6
 - setting high water marks 18-14
 - size, guidelines 18-6
 - size, performance considerations 18-5
 - See also* Logical-log buffer, Logical-log file
- Logical page number 40-34
- Logical recovery
 - number of threads 35-32
 - role in data replication 25-9
- Logical units of storage
 - description of 10-14

- list of 10-4
- Logical volume manager (LVM)
 - description of 10-44
 - mirroring alternative 23-6
- Logical-log buffer
 - and data-replication buffer 14-25
 - and LOGBUFF parameter 35-20
 - and logical-log records 14-44
 - description of 14-24
 - monitoring 29-41
 - number of 14-24
 - role in logging process 16-8
 - synchronizing flushing 14-39
 - tuning size for performance 30-5, 30-12
 - when flushed to disk 14-44, 18-20
- Logical-log file
 - adding a log file
 - using ON-Monitor 19-4
 - using onparams 19-4, 19-5
 - allocating disk space for 18-5
 - and reserved pages 40-8
 - backup
 - effect on performance 18-5
 - goals of 18-10
 - to free deleted blobpages 18-16
 - changing the size of 19-7
 - consequences of not freeing 18-11
 - created during initialization 35-20
 - description of 18-4
 - displaying contents 37-23
 - dropping a log file
 - using ON-Monitor 19-6
 - using onparams 19-6
 - file status 18-9
 - how to free 19-10
 - how to switch 19-12
 - I/O to 12-21
 - location 18-8
 - logid number 18-8
 - mirroring a dbspace containing a file 23-6
 - moving to another dbspace 19-6
 - number of files 18-7
 - rate at which files fill 18-13
 - reading the log file 37-23
 - relationship between unique id and logid 18-9
 - reserved page information 40-8
 - role in fast recovery 22-5, 22-7 to 22-8
 - size 18-7, 35-21
 - status flags 18-9
 - switch using onmode 37-31
 - switching to activate blobspaces
 - chunks 18-16
 - switching to activate blobspaces 18-15
 - unique id number 18-8
 - when freed for re-use 18-11
 - when OnLine tries to free files 18-11
 - See also* Logical log
- Logical-log I/O virtual processors 12-21
- Logical-log record
 - additional columns of 39-7
 - displaying 37-23
 - flushing under two-phase commit protocol 32-29
 - for a checkpoint 39-4
 - for a drop table operation 39-4
 - generated by a rollback 39-4
 - header columns 39-6
 - involved in distributed transactions 39-5
 - involved in two-phase commit protocol 32-28, 39-5
 - OnLine processes requiring 16-4
 - role in fast recovery 22-6, 22-7 to 22-8
 - role in two-phase commit protocol 32-6
 - SQL statements generating 16-6
 - types 39-7
 - when written to logical-log buffer 18-19
- logid 18-8
- LOGSIZE parameter
 - changing 19-8
 - description of 35-21
 - mentioned 3-21
 - use in logical log size determination 18-7
- LOGSMAX parameter
 - changing 19-9
 - description of 35-22
 - mentioned 3-21, 18-7
- Long transaction
 - consequences of 18-12
 - description of 18-12
 - mentioned 3-22
 - mentioned within two-phase commit discussion 32-18, 32-22, 32-25
 - preventing development of 18-13
 - starting percentage 35-25

- LRU queues
 - and buffer pool management 14-33
 - and buffer table 14-23
 - components 14-32
 - description of 14-32
 - displaying information with onstat 37-64
 - FLRU queues 14-32
 - MLRU queues 14-32
 - modified pages, percentage of 35-22, 35-23
 - tuning parameters for performance 30-13
 - LRU write
 - description of 14-43
 - monitoring 29-20
 - LRUS parameter
 - description of 14-32, 35-22
 - mentioned 3-24
 - tuning for performance 30-14
 - LRU_MAX_DIRTY parameter and LRU_MIN_DIRTY parameter 14-32
 - description of 35-22
 - example of use 14-34
 - how to calculate value 14-34
 - mentioned 3-24
 - purpose of 14-34
 - role in buffer pool management 14-34
 - tuning for performance 30-14
 - LRU_MIN_DIRTY parameter and LRU_MAX_DIRTY parameter 14-32
 - default value 14-34
 - description of 35-23
 - example of use 14-35
 - how to calculate value 14-35
 - mentioned 3-25
 - role in buffer pool management 14-34
 - tuning for performance 30-14
 - when tested 14-42
 - LTAPEBLK parameter
 - description of 35-23
 - mentioned 3-23
 - LTAPEDEV parameter
 - description of 35-24
 - in a learning environment 3-14
 - in a production environment 3-19
 - mentioned 3-23, 3-26
 - with onunload/onload 31-7
 - LTAPESIZE parameter
 - changing tape size 35-24
 - description of 35-24
 - mentioned 3-23
 - LTXEHWM parameter
 - and physical log 20-7
 - changing 19-9
 - description of 35-25
 - mentioned 3-21
 - role in heuristic rollback 32-21
 - role in preventing long transactions 18-15
 - LTXHWM parameter
 - changing 19-9
 - description of 35-25
 - mentioned 3-21
 - role in heuristic rollback 32-21
 - role in preventing long transactions 18-14
- ## M
- Machine notes Intro-9, 9
 - Main_loop thread
 - role in checkpoint 14-49
 - role in two-phase commit recovery 32-11
 - Managing physical disk I/O 10-3
 - Manual recovery
 - deciding if action needed 33-8
 - determining if data inconsistent 33-5
 - example of 33-9
 - obtaining information from logical log files 33-6
 - procedure to determine if necessary 33-3
 - use of GTRID 33-6
 - Mapping, bad sector 1-10
 - Media failure
 - detecting 23-9
 - recovering from 23-5
 - restoring data 1-7
 - Memory. *See* Shared memory
 - Message area, communications portion (shared memory) 14-29
 - Message file for error messages Intro-9, 9
 - Message log
 - alphabetical listing of messages 38-3
 - and data corruption 27-8
 - categories of messages 38-4

-
- description of 29-6
 - displaying with onstat utility 37-61
 - during initialization 9-9
 - file pathname 35-27
 - location of 35-27
 - mentioned 42-6
 - monitoring 29-7
- Message segments of shared memory,
mentioned 9-6
- Migration
- of data using NLS 31-17
 - onload utility 37-18
 - onunload utility 37-77
 - See also* Moving data
- Mirror chunk
- adding 24-8
 - changing status of 24-8
 - creating 24-5
 - disk reads from 23-8
 - disk writes to 23-8
 - pathnames 40-11
 - recovering 23-9, 24-8
 - relinking after disk failure 24-10
 - structure 23-10, 40-15
- MIRROR parameter
- changing 24-4
 - description of 24-4, 35-26
 - initial configuration value 24-4
 - mentioned 3-21
- Mirroring
- activity during processing 23-8
 - alternatives 23-5
 - and chunk table 14-19
 - asynchronous write requests 23-8
 - benefits of 23-4
 - changing chunk status 24-8
 - costs of 23-4
 - creating mirror chunks 24-5
 - description of 1-7, 23-3
 - detecting media failures 23-9
 - effects of 3-21
 - enable flag 35-26
 - enabling 24-4
 - ending 24-10
 - if the dbospace holds logical-log files 23-6
 - initial chunk 35-27
 - network restriction 23-4
 - recommended disk layout 10-31
 - recovering a chunk 24-8
 - recovery activity 23-7
 - reserved page information 40-11
 - sample disk layout 10-37
 - split reads 23-8
 - starting 24-3, 24-5
 - status flags 23-7
 - steps required 24-3
 - what happens during processing 23-8
 - when mirroring begins 23-6
 - when mirroring ends 23-10
- MIRROROFFSET parameter 10-17
- description of 35-26
 - mentioned 3-21
 - setting 24-6
 - when needed 11-6
- MIRRORPATH parameter
- description of 35-27
 - mentioned 3-21, 10-17
 - multiple residency 6-6
 - setting 24-6
 - specifying a link pathname 35-27
- MLRU queues
- and flushing of regular buffers 14-39
 - and LRU_MIN_DIRTY parameter 14-34
 - description of 14-32
 - how buffer is placed 14-34
 - how to end page-cleaning 14-35
 - limiting number of pages 14-34
 - role in buffer modification 14-38
 - when cleaning ends 14-34
 - See also* LRU queues.
- Mode
- current operating mode 8-4
 - description of 7-3
 - graceful shutdown 8-5
 - immediate shutdown 8-5
 - list of mode changes 8-3
 - list of OnLine modes 7-3
 - off-line from any mode 8-6
 - 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 8-5
 - quiescent 7-3
 - quiescent to on-line 8-4
 - reinitializing shared memory 8-3
 - taking off-line 8-6
 - users permitted to change 8-3

-
- MODE ANSI keywords, and database
 - logging status 16-9
 - Monitoring OnLine
 - active tablespaces 29-26
 - blobs in blobspaces 29-53
 - blobs in dbspaces 29-56
 - blob space storage efficiency 11-17
 - buffer-pool activity 29-19
 - buffers 14-23, 29-15
 - caching percentages 30-4
 - checkpoints 29-11
 - chunk status 29-43
 - chunks 11-19, 29-46
 - configuration parameter values 29-9
 - databases 29-36
 - data-replication status 29-58
 - disk I/O queues 12-23
 - extents 29-50
 - latches 29-12, 29-22
 - list of tools 3-5
 - list of topics 29-5
 - locks 29-23
 - logging status 29-36
 - logical-log buffers 29-41
 - logical-log files 29-37
 - message log 29-7
 - page-cleaning activity 30-14
 - pages written per I/O 30-5
 - physical-log buffer 14-25, 29-41
 - physical-log file 29-40
 - profile of activity 29-13
 - sessions 29-29
 - shared memory 29-12, 29-13
 - shared-memory segments 29-12
 - sources of information 29-6
 - stack size 29-30
 - threads 29-29
 - transactions 29-33
 - user threads 14-23
 - using ON-Monitor 29-8
 - using SMI tables 29-8
 - virtual processors 29-27
 - Moving data
 - between locales 31-17
 - blobspaces 31-8
 - CHAR data type with NLS 31-17
 - choosing between dbload, dbimport, and LOAD 31-8
 - configuration parameters 3-26
 - constraints 31-5
 - conversion of locale 31-18
 - creating a database object 31-9
 - dbexport utility 31-13
 - dbimport utility 31-13
 - diagram of choosing a method 31-8
 - differences between OnLine and SE 31-16
 - fixed format 31-12
 - from external source 31-12
 - from OnLine to INFORMIX-SE 31-14
 - DBPATH variable 31-14
 - from SE to OnLine 31-16
 - from SE, schema file 31-16
 - ignoring records 31-11
 - level-0 archive required 31-11
 - modifying the schema 31-10
 - NCHAR data type 31-17
 - oncheck utility 31-8, 31-11
 - overview 31-3
 - privileges required 31-13
 - rearrange input data 31-11
 - reasons for moving 31-4
 - skipping bad records 31-11
 - steps for moving NLS data 31-18
 - steps for using onunload/onload 31-7
 - steps for using UNLOAD/LOAD 31-11
 - summary of methods 31-4
 - VARCHAR data type, with NLS 31-17
 - MSGPATH parameter
 - description of 35-27
 - in a learning environment 3-15
 - in a production environment 3-19
 - mentioned 3-23
 - multiple residency 6-5
 - Multimedia support 1-8
 - Multiple concurrent threads (MCT) 12-10
 - Multiple connection types
 - example 4-23
 - in sqlhosts file 4-18
 - See also* Connection.
 - Multiple instances of OnLine on same computer 5-3
 - Multiple network interface cards 12-30
 - Multiple residency
 - and blob space 6-6
 - and chunk assignment 6-6
 - and dbspaces 6-6
 - and multiple binaries, warning 6-4
 - archiving 6-7
 - backups 6-7
 - benefits of 5-3

- configuration and setup 6-3
 - configuration file 5-3
 - configuration parameters 5-4
 - DBSERVERNAME parameter 6-5
 - definition of 3-10, 5-3
 - editing the ONCONFIG file 6-5
 - example 4-25
 - how it functions 5-4
 - INFORMIXSERVER environment variable 6-8
 - informix.rc & .informix files 6-8
 - initializing disk space 6-6
 - IPX/SPX 6-6
 - MIRRORPATH parameter 6-6
 - MSGPATH parameter 6-5
 - ONCONFIG environment variable 5-4
 - planning for 6-3
 - ROOTNAME parameter 6-5
 - ROOTPATH parameter 6-5
 - SERVERNUM parameter 6-4, 6-5
 - sqlhosts file 6-6
 - startup script 6-8
 - steps for preparing 6-4
 - to isolate applications 5-3
 - use for testing 5-4
 - /etc/hosts & /etc/services 6-6
 - Multiprocessor computer
 - advantages on 12-3
 - AFF_SPROC parameter 35-8
 - MULTIPROCESSOR parameter 12-17
 - processor affinity 12-9
 - two-processor machine 12-17
 - MULTIPROCESSOR parameter
 - description of 12-17, 35-27
 - for single-processor computer 12-17
 - Multi-threaded database server. *See*
 - Dynamic scalable architecture
 - Multithreaded processes, description of 12-5
 - Mutex
 - description of 12-16, 14-30
 - on buffer table hash table 14-36
 - when used 14-30
- N
- Native Language Support
 - environment variables 3-18
 - mentioned 3-27
 - steps for moving NLS database 31-17
 - NCHAR data type, changing locale 31-17
 - nettype field
 - description of 4-12
 - format of 4-12
 - summary of values 4-14
 - syntax of 4-12
 - use of interface type 4-22
 - NETTYPE parameter
 - and communications portion (shared memory) 14-30
 - and multiple network addresses 12-29
 - description of 35-28
 - mentioned 4-17
 - ON-Monitor screen entries 13-5
 - poll threads 12-25
 - purpose of 4-17
 - role in specifying a protocol 12-25
 - tuning example 35-29
 - vp class entry 12-25
 - Netware file server 4-15
 - Network administration 4-7
 - Network communication
 - connection types 12-24
 - using IPX/SPX 4-15, 4-16, 4-23
 - using TCP/IP 4-15, 4-16
 - Network configuration files 4-7
 - Network connection
 - how implemented 12-26
 - types of 12-24
 - when to use 4-5
 - Network Information Service 4-8
 - Network interface cards
 - and listen threads 12-30
 - sqlhosts file 12-30
 - using multiple 12-30
 - Network interface, definition 4-13
 - Network protocols
 - specifying 12-25
 - Network security
 - files 4-9
 - .rhosts file 4-9
 - /etc/hosts.equiv 4-9
 - /etc/passwd 4-9
 - /etc/shadow 4-9
 - Network virtual processors
 - and poll threads 12-25
 - description of 12-24
 - how many 12-25
 - performance of 30-20

- NIS server 4-9
 - NIS servers, effect on /etc/hosts and /etc/services 4-9
 - NOAGE parameter
 - description of 35-30
 - purpose of 12-18
 - recommendation 30-19
 - Node, index (disk)
 - branch node defined 40-45
 - checking horizontal and vertical nodes 37-10
 - contents of leaf nodes 40-47
 - creation of branch nodes 40-47
 - defined 40-44
 - leaf node defined 40-45
 - pointer 40-47
 - root node 40-45
 - root node defined 40-45
 - types of 40-45
 - what branch nodes point to 40-49
 - when root node fills 40-46
 - Non-Informix databases 1-8
 - NUMAIOVPS parameter
 - description of 35-30
 - purpose of 12-23
 - Number of
 - dbspaces 35-13
 - page-cleaner threads 35-11
 - NUMCPUVPS parameter
 - and poll threads 12-26
 - and SINGLE_CPU_VP parameter 12-18
 - description of 35-31
 - purpose of 12-16
 - NVARCHAR data type, changing locale 31-17
- O**
- Off-line mode 7-3
 - Offset
 - definition of 10-8
 - purpose of 11-5
 - use in prevention of overwriting partition information 10-8
 - use in subdividing partitions 10-8
 - when needed 11-5
 - OFF_RECVRY_THREADS parameter
 - description of 35-31
 - ON-Archive
 - backing up logical-log files 18-11, 18-12
 - catalog tables 36-3
 - configuration file 3-26
 - config.arc file 3-26
 - mentioned 3-23
 - modifying database logging status 17-5
 - use in starting data replication 26-9
 - onaudit utility
 - and ADTERR parameter 35-6
 - and ADTMODE parameter 35-7
 - mentioned 35-7
 - oncatlgr utility
 - in UNIX startup script 3-29
 - message file 42-6
 - multiple residency 6-7
 - oncatlgr.out.pidnum file 42-6
 - oncfg_servername.servernum file 9-8, 42-8
 - oncheck utility
 - before moving data 31-8, 31-11
 - before moving NLS data 31-17, 31-18
 - before using dbexport/dbimport 31-13
 - blob storage information 11-18
 - check-and-repair options 37-6
 - comparison with onstat 29-8
 - corrective actions 27-4
 - description of 37-6
 - list of functions 37-7
 - obtaining blobstorage information 29-55
 - obtaining blob-storage statistics 29-54
 - obtaining chunk information 29-44, 29-48
 - obtaining configuration information 29-10
 - obtaining extent information 29-50
 - obtaining logical-log information 29-38
 - obtaining physical-log information 29-41
 - obtaining tblspace information 29-52, 29-56
 - options
 - cc 37-9
 - cd 37-9
 - ce 37-10
 - cl 37-11

- ci 37-10
- cr 37-11
- n 37-12
- pB 11-17, 37-12
- pc 37-12
- pD 37-12
- pd 37-12
- pe 37-13, 40-4
- pK 37-13
- pk 37-13
- pL 37-13
- pl 37-13
- pP 37-14
- pp 37-13
- pr 37-14
- pT 37-14
- pt 37-14
- q 37-15
- y 37-15
- overview of functionality 37-6
- syntax 37-8
- use in consistency checking 27-4
- ONCONFIG configuration file
 - conventions used 35-5
 - description 35-6, 42-7
 - during initialization 9-5, 9-6
 - editing with multiple residency 6-5
 - for a learning environment 3-13
 - mentioned 4-17
 - multiple residency 4-25, 6-4
 - parameters 4-17
 - relation to INFORMIXSERVER 3-9
 - whitespace 35-6
- ONCONFIG environment variable
 - and ONCONFIG file 42-7
 - changes for multiple residency 6-4
 - interaction with TBCONFIG 42-7
 - mentioned 3-9
 - multiple residency startup script 6-8
 - multiple versions of OnLine 3-29
 - use with multiple residency 5-4
- ONCONFIG parameter
 - conventions 35-6
- onconfig.std file
 - and multiple residency 6-4
 - description 42-7
 - during initialization 9-6
 - in a learning environment 3-13
 - sample 42-10
 - when installed 3-8
- oninit utility
 - bringing OnLine on-line 8-4, 11-7
 - initializing OnLine 8-4
 - option descriptions 37-17
 - p option 9-8
 - starting OnLine 37-16
 - temporary tables 37-17
- On-line files
 - documentation notes Intro-8, 8
 - machine notes Intro-9, 9
 - provided with the product Intro-8, 8
 - release notes Intro-9, 9
- On-line mode, description of 7-4
- Online, see INFORMIX-OnLine
 - Dynamic Server
- onload utility
 - constraints 37-21
 - create options 37-20
 - description of 37-18
 - mentioned 31-4
 - specifying tape parameters 37-19
 - syntax 37-18
 - using 31-5, 31-7
 - See also* onunload utility
- onlog utility
 - description of 37-23
 - filters for displaying logical-log
 - records 37-25
 - filters for reading logical-log records
 - 37-24
 - reconstructing a global transaction
 - 33-6
- onmode utility
 - adding a shared-memory segment
 - 37-34
 - bringing OnLine on-line 8-4
 - changing OnLine mode 37-29
 - changing shared-memory residency
 - 15-15, 37-30
 - description of 37-27
 - dropping CPU virtual processors 13-9
 - forcing a checkpoint 15-16, 37-30
 - freeing a logical-log file 19-11, 19-12
 - graceful shutdown 8-5
 - immediate shutdown 8-6
 - killing a participant thread 32-18
 - killing a session 32-21, 37-31
 - killing a transaction 32-18, 32-26, 33-4,
 - 37-32
 - options

- a add shared memory segment 37-34
 - b change database format 37-35
 - c force checkpoint 37-30
 - d set data replication type 37-32
 - k take off-line 37-29
 - l switch logical-log file 37-31
 - m quiescent to on-line 37-29
 - n end forced residency 37-30
 - p add or remove virtual processor 37-34
 - r begin forced residency 37-30
 - R regenerate .infos file 37-36
 - s take to quiescent 37-29
 - u immediately to quiescent 37-29
 - z kill client session 37-31
 - Z kill transaction 37-32
 - setting database server type 26-9, 26-17, 37-32
 - switching logical-log files 19-12, 37-31
 - take OnLine off-line 8-7
 - user thread services onmode utility requests 12-5
- ON-Monitor**
- access and use 34-3
 - adding a logical-log file 19-4
 - adding mirror chunks 24-8
 - archive menu and options 34-10
 - data-replication screen 34-15
 - dbspaces menu and options 34-7
 - diagnostics screen 34-16
 - dropping a logical-log file 19-6
 - enabling mirroring 24-4
 - ending mirroring 24-10
 - force-ckpt option 34-9
 - help 34-4
 - initialization screen 34-12
 - logical-logs menu and options 34-11
 - mode menu and options 34-8
 - modifying database logging status 17-8
 - parameters menu and options 34-6
 - recovering a chunk 24-9
 - setting performance options 15-12
 - setting shared memory parameters 15-7, 15-9
 - setting virtual processor parameters 13-3
 - shared-memory screen 34-13
 - starting mirroring 24-7
 - status menu and options 34-5
 - taking a chunk down 24-9
 - virtual processor screen 34-14
- onparams utility**
- adding a logical-log file 19-4, 37-38
 - change physical log size, location 37-39
 - changing physical-log location 21-5
 - changing physical-log size 21-5
 - description of 37-37
 - dropping a logical-log file 19-6, 37-38
- onspaces utility**
- adding a chunk 37-42
 - adding a mirror chunk 24-8
 - changing chunk status 37-45
 - creating a blobspace or dbspace 37-41
 - description of 37-40
 - dropping a blobspace or dbspace 37-42
 - dropping a chunk 37-43
 - ending mirroring 24-11, 37-45
 - recovering a down chunk 24-9, 37-45
 - starting mirroring 24-7, 37-44
 - taking a chunk down 24-9
- onstat utility**
- and CPU virtual processors 12-17
 - comparison with oncheck utility 29-8
 - description of 37-46
 - freeing blobpages and timing 37-53
 - header 37-48
 - monitoring blobspace 29-53
 - monitoring buffer use 14-23, 29-15, 29-16, 29-17, 29-18
 - monitoring buffer-pool 29-20, 29-21
 - monitoring byte locks 14-21
 - monitoring checkpoints 29-11
 - monitoring chunk status 29-44
 - monitoring configuration 29-9
 - monitoring data replication 29-58
 - monitoring disk usage 29-47
 - monitoring latches 29-22, 29-23
 - monitoring locks 29-24
 - monitoring log buffers 29-41
 - monitoring logical-log files 29-37
 - monitoring OnLine profile 29-13, 29-14
 - monitoring physical log 29-40
 - monitoring sessions 29-29
 - monitoring shared memory 29-12, 29-13
 - monitoring tblspaces 29-26, 29-28
 - monitoring the message log 29-7
 - monitoring transactions 29-33

- monitoring virtual processors 29-27,
29-28
 - option descriptions 37-49
 - options
 - a 37-49
 - B 37-51
 - b 37-50
 - c 37-51
 - D 37-54
 - d 10-16, 37-52, 37-53
 - F 37-54
 - k 37-58
 - l 37-59
 - m 37-61
 - o 37-61
 - p 37-61
 - R 37-64
 - r 37-63
 - s 37-64
 - t 37-65
 - u 37-66
 - X 37-69
 - z 37-69
 - 37-49
 - (none) 37-49
 - repeated execution with -r 37-63
 - repeated execution with seconds
parameter 37-48
 - syntax 37-47
 - table of options and functions 37-46
 - terminating interactive mode 37-58
 - terminating repeating sequence 37-58
 - tracking a global transaction 32-27
 - using SMI tables for onstat
information 36-24
 - using with shared-memory source file
37-48
 - ontape utility
 - archiving an OnLine database server
37-72
 - backing up logical-log files 18-10,
18-11, 37-74
 - changing database logging status
37-73
 - data replication functions 37-76
 - description of 37-70
 - device pathname 35-24
 - exit codes 37-71
 - LTAPEBLK, use of 35-23
 - mentioned 3-23
 - modifying database logging status
17-6
 - restoring data from an archive 37-75
 - starting continuous backup 37-74
 - tasks performed by 37-70
 - use in starting data replication 26-9
 - onunload utility
 - description of 37-77
 - locking 37-79
 - logging mode 37-79
 - LTAPEBLK, use of 35-23
 - mentioned 31-4
 - ownership and privileges 37-78
 - steps for using 31-7
 - syntax 37-77
 - use without tape parameters 37-78
 - using 31-5
 - what is included with a database
37-79
 - what is included with a table 37-79
 - onunload utility *See also* onload utility
 - On_Archive
 - fatal error record 42-9
 - ON_RECVRY_THREADS parameter
 - description of 35-32
 - role in data replication 25-13
 - Operating OnLine, things to avoid 28-3
 - Operating system files. *See* Cooked file
space
 - oper_deflt.arc file
 - and multiple residency 6-7
 - contents 42-8
 - Optical storage
 - and STAGEBLOB parameter 35-41
 - data types for 1-8
 - Optical (OPT) virtual processor 12-30
 - ovbuff field, performance tuning 30-5
 - ovlock field, performance tuning 30-5
 - ovtbls field, performance tuning 30-5
 - ovuser field, performance tuning 30-5
- ## P
- Page
 - bit-map page 40-59
 - blobpage blobpage 40-59
 - blobpage free-map page 40-59
 - components of dbspace page 40-30
 - compression 40-33, 40-41
 - dbspace blob page 40-57
 - dbspace page types 40-25

- definition of full page 40-35
- description of 10-9
- determining OnLine page size 11-13
- free page, definition of 40-25
- fullness bit values 40-23
- fullness, 4-bit values 40-23
- header components 40-31
- locating in shared memory 14-36
- logical page number 40-34
- page types in extent 40-25
- physical page number 40-34
- relationship with chunk 10-9
- size recorded in reserved pages 40-7
- slot table 40-32
- structure and storage of 40-30
- validating consistency 27-5
- Page compression 40-32, 40-33, 40-41
- Page-cleaner table
 - description of 14-21
 - number of entries 14-21
- Page-cleaner threads
 - and LRU writes 14-43
 - codes for activity state 37-54
 - description of 14-39
 - efficiency trade-offs 30-14
 - flushing buffer pool 14-39
 - flushing of regular buffers 14-39
 - monitoring 14-21
 - monitoring activity 37-54
 - number of 35-11
 - role in chunk write 14-43
 - sleeping forever 14-42
 - tuning parameters for performance 30-13
- PAGE_1CKPT reserved page 40-6
- PAGE_1PCHUNK reserved page 40-10
- PAGE_2CKPT reserved page 40-6
- PAGE_ARCH reserved page 40-7, 40-12
- PAGE_CKPT reserved page 40-8
- PAGE_CONFIG reserved page 37-11, 40-8
 - mentioned 9-6, 9-8
- PAGE_DBSP reserved page 40-9
- PAGE_MCHUNK reserved page 40-11
- PAGE_PCHUNK reserved page 40-10
- PAGE_PZERO reserved page
 - contents of 40-7
 - mentioned 9-7
 - when written to 40-6
- Parallel processing
 - mentioned 1-6
 - PSORT_NPROCS environment variable 30-8
 - sorting 30-8
 - virtual processors 12-8
- Participant database server
 - automatic recovery 32-14
 - description of 32-5
- Partnum field in systables 40-19
- PATH environment variable
 - in shutdown script 3-29
 - in startup script 3-28
 - mentioned 3-9
 - multiple residency startup script 6-8
- Pathname for ontape, onunload & onload 35-24
- Pending transaction 37-68
- Performance
 - advantages of raw-disk management 1-5
 - and shared memory 14-5
 - effect of read-ahead 14-35
 - how frequently buffers are flushed 14-32
 - of CPU virtual processors 12-17
 - shared-memory connection 4-4
- Performance configuration parameters
 - setting, using a text editor 15-11
 - setting, using ON-Monitor 15-12
- Performance tuning
 - and extent size 10-34
 - and logical volume managers 10-44
 - blobospace blobpage size 11-16
 - checkpoint frequency 30-16
 - disk layout guidelines 10-30
 - log buffer sizes 30-12
 - logical-log size 18-5
 - mechanisms 1-5
 - minimizing disk head movement 10-33
 - moving the physical log 21-3
 - page-cleaner parameters 30-13
 - reducing disk contention 10-33
 - shared-memory buffers 30-11
 - shared-memory resources 30-17
 - specifying sorting directory 30-7
 - spreading data across multiple disks 10-44
 - tuning amount of data logged 20-4

- user guidelines 30-10
 - when needed 30-4
- Permissions, file 3-13, 11-4
- PHYSBUFF parameter
 - and physical-log buffers 14-25
 - description of 35-32
 - mentioned 3-22
- PHYSDBS parameter
 - changing size and location 21-5
 - description of 35-33
 - mentioned 3-22
 - where located 20-7
- PHYSFILE parameter
 - changing size and location 21-5
 - description of 35-33
 - initial configuration value 20-6
 - mentioned 3-22
- Physical consistency, description of 22-5
- Physical log
 - and virtual processors 12-21
 - becoming full 20-6
 - before-image contents 20-4
 - buffer 20-8
 - changing size and location
 - possible methods 21-3
 - rationale 21-3
 - restrictions 21-4
 - using an editor 21-5
 - using ON-Monitor 21-4
 - using onparams 21-5
 - checking consistency 37-11
 - configuration parameters for 3-22
 - description of 20-5
 - effects of checkpoints on sizing 20-6
 - effects of frequent updating 20-5
 - ensuring does not become full 20-6
 - flushing of buffer 20-9
 - how emptied 20-9
 - in root dbspace 40-4
 - I/O, virtual processors 12-22
 - monitoring 29-40
 - optimal storage of 10-35, 10-36
 - role in fast recovery 22-4, 22-5, 22-5
 - scenario for filling 20-7
 - size of 35-33
 - sizing guidelines 20-5
 - where located 20-7
- Physical logging
 - and archiving 20-4
 - and blobs 20-4
 - and data buffer 20-9
 - and fast recovery 20-4
 - description of 20-3
 - details of logging process 20-8
 - purpose of 20-3
 - which activity logged 20-4
- Physical page number 40-34
- Physical units of storage
 - description of 10-4
 - list of 10-3
- Physical-log buffer
 - amount written 14-41
 - and checkpoints 20-9
 - dbspace location 35-33
 - description of 14-25
 - events that prompt flushing 14-40
 - flushing of 14-39, 20-9
 - monitoring 29-41
 - number of 14-25
 - PHYSBUFF parameter 14-25
 - role in dbspace logging 18-19, 20-8
 - size of 35-32
 - tuning size for performance 30-5, 30-12
 - when it becomes full 14-40
- PIO virtual processors
 - description of 12-22
 - how many 12-22
- Planning for OnLine resources 3-4
- Platforms with more than two CPUs 12-17
- Poll threads
 - and message queues 14-29
 - DBSERVERNAME parameter 12-25
 - description of 12-26
 - how many 12-25
 - in NETTYPE parameter 35-28
 - multiple for a protocol 12-25
 - nettype entry 12-25
 - on CPU or network virtual processors 12-25
- Post-decision phase 32-6, 32-9
- Practice database, preparing 3-17
- Precommit phase 32-6
- Preparation
 - of production environment 3-17
- Preparation of
 - cooked disk space 3-13
 - ONCONFIG file 3-13
 - sqlhosts file 3-15

-
- Presumed-abort optimization 32-10, 32-17
 - Primary database server 25-4
 - Priorities for disk I/O 12-21
 - Priority aging
 - description of 12-18
 - of CPU virtual processors 35-30
 - Private environment file 42-5
 - Privileges
 - access privileges for tables 37-79
 - database schema 31-10
 - for the demonstration database
 - Intro-11
 - migrating with onunload 31-8
 - on databases and tables 1-8
 - required for dbexport 31-13
 - required for onload 37-22
 - required with onunload 37-78
 - with onunload 37-21
 - Procedure Name segment 31-15
 - Processes
 - compared to threads 12-4
 - that attach to shared-memory 14-10
 - Processor affinity
 - AFF_NPROCS parameter 35-8
 - and AFF_SPROC parameter 35-8
 - description of 12-9
 - using 12-18
 - Processor, locking for multiple or single 35-27
 - Production environment, configuration 3-17
 - Profile
 - displaying counts with onstat utility 37-61
 - monitoring with SMI 36-15
 - setting counts to zero 37-69
 - Program counter and thread data 14-27
 - Protocol
 - in NETTYPE parameter 35-28
 - specifying 12-25
 - PSORT_DBTEMP environment variable
 - creating temporary implicit tables 10-25
 - listing directories for intermediate writes 30-10
 - relationship to DBSPACETEMP 35-14
 - PSORT_NPROCS environment variable
 - allocating sort memory 14-29
 - to enable Psort package 30-9
- ## Q
- Queues
 - description of 12-14
 - ready 12-14
 - sleep 12-14
 - wait 12-15
 - Quiescent mode
 - description of 7-3
 - with oninit utility 37-17
- ## R
- RAID. *See* Redundant array of inexpensive disks.
 - Railroad diagrams
 - conventions used in Intro-5, 5
 - example of syntax conventions Intro-7, 7
 - Raw device 10-6
 - and character-special interface 10-6
 - definition of 10-6
 - Raw disk space 10-6
 - compared with cooked space 10-6
 - definition 3-12
 - description of 10-5
 - how to allocate 11-5
 - in data storage 10-3
 - rationale for using 10-7
 - steps for allocating 10-7
 - Raw-disk management 1-5
 - RA_PAGES parameter
 - description of 35-34
 - purpose of 14-35
 - reading a page from disk 14-37
 - RA_THRESHOLD parameter
 - description of 35-34
 - purpose of 14-35
 - RDBMS 1-3
 - Read-ahead
 - description of 14-35
 - number of pages 35-34
 - RA_PAGES parameter 14-35
 - RA_THRESHOLD parameter 14-35
 - threshold for 35-34
 - using onstat to monitor 14-36
 - when it occurs 14-37
 - when used 14-35

-
- Read-only mode
 - description of 7-4
 - Ready queue
 - description of 12-14
 - moving a thread to 12-14, 12-15
 - Reception buffer 25-9
 - Recovery
 - by two-phase commit protocol 32-10
 - fast, description of 22-3
 - from media failure 23-5
 - Recovery mode, description of 7-4
 - Recovery threads
 - off-line 35-31
 - on-line 35-32
 - Redundant array of inexpensive disks (RAID)
 - mirroring alternative 23-6
 - Referential constraints 16-4
 - Regular buffers
 - and big buffers 14-27
 - description of 14-23
 - events that prompt flushing 14-39
 - how big 14-24
 - monitoring status of 14-23
 - Relational database management system (RDBMS) 1-3
 - Relay Module, version 5.0
 - example 4-29
 - Relay module, version 5.0
 - description of 4-29
 - Relay module, version 6.0
 - description of 4-26
 - example 4-26
 - example with three servers 4-28
 - Release notes Intro-9, 9
 - Remainder page, description of 40-36
 - Remote client 4-9
 - Remote computer 4-9
 - Remote hosts 4-9
 - Replication server
 - See* Data replication.
 - Reserved pages
 - archive information 40-7
 - checking with oncheck 37-11
 - checkpoint information 40-6
 - data-replication information 40-7
 - dbspace information 40-7
 - description of 40-6
 - location in root dbspace 40-4
 - optimal storage 10-35
 - organization in pairs 40-6
 - role in checkpoint processing 40-8
 - validating with oncheck 27-4
 - viewing of contents 40-7
 - when updated 40-6
 - RESIDENT parameter
 - changing 15-15
 - description of 35-35
 - during initialization 9-9
 - mentioned 3-25
 - Resident segment of shared memory 9-6
 - Resident shared memory
 - setting configuration parameters 15-7
 - turning on/off residency 15-15
 - Resource planning for OnLine 3-4
 - Revert database to 5.0 format 37-35
 - Roll back
 - in fast recovery 22-8
 - mentioned 1-7
 - Roll forward
 - in fast recovery 22-7
 - mentioned 1-7
 - Root dbspace
 - and temporary tables 10-17
 - calculating size of 10-27
 - description of 10-17
 - disk layout
 - for production environment 10-30
 - initial chunk 35-36
 - location of logical-log files 18-8
 - mentioned 3-20
 - mirroring 24-6, 35-27
 - specified by ROOTNAME parameter 35-35
 - structure 40-4
 - using a link 35-36
 - ROOTNAME parameter
 - description of 35-35
 - mentioned 3-20, 10-17
 - multiple residency 6-5
 - relationship to DBSPACETEMP 35-13
 - used by PHYSDBS 35-33
 - ROOTOFFSET parameter
 - description of 35-36
 - mentioned 3-20, 10-17
 - multiple residency 6-5
 - when is it needed 11-6

ROOTPATH parameter
description of 35-36
in a production environment 3-19
in learning environment 3-14
mentioned 3-20, 10-17
multiple residency 6-5
specifying as a link 35-36

ROOTSIZE parameter
description of 35-36
mentioned 3-20

Row
accomodating large rows 40-39
data row storage 40-35
displaying contents with oncheck 37-12
effects of deletion on index 40-52
effects of modifying 40-39
linking of sections 40-36
maximum in a page 40-32
storage location 40-36

Rowid
and SMI 36-5
as component of index item 40-44
description of 40-33
effect of page compression 40-33
effects of changing 40-33
elements of 40-32
format 40-34
functions as forward pointer 40-34
locking information derived from 37-59
relation to slot table 40-32
stored in index pages 40-34
structure 40-33
where stored 40-34

RSAM task control block 29-29

S

Sample onconfig.std file 42-10

Scans
of indexes 14-35
of sequential tables 14-35

Schema file
created by dbexport 31-13
definition 31-10
modify when moving data 31-13
moving data from SE 31-16

Secondary database server 25-4

Security
C2-level secure auditing Intro-13
how enforced 1-8
isolating applications 5-3
network 4-9
of database server 1-8
risks with shared-memory communications 4-4

Segment
See Chunk. 17

Segment identifier (shared-memory) 14-12

Semaphore, UNIX parameters 15-5

SERVERTNUM parameter
and calculating key value 14-12
and multiple OnLine's 14-12
and multiple residency 5-4, 6-4, 6-5
description of 35-37
how used 14-11
in a learning environment 3-15
in a production environment 3-19
mentioned 3-20

servicename field in sqlhosts file
choosing an appropriate name 4-15
description of 4-15
with IPX/SPX 4-16
with shared memory 4-16

Service, in IPX/SPX 4-16

Session
and active tbspace 14-21
and dictionary cache 14-28
and locks 14-21
and shared memory 14-27
and stored procedure cache 14-29
control block 12-10
description of 12-10
information in SMI tables 36-16, 36-17
monitoring 29-29
primary thread 14-27
shared-memory pool 14-26
sqlxec threads 12-5
stack and heap 14-26
threads 12-5

Session control block 12-10
description of 14-27
shared memory 14-27

SET CONSTRAINTS statement 31-14
 SET ISOLATION statement 31-14
 SET LOCK MODE TO statement 31-15
 SET LOG statement 31-14
 Share lock (buffer), description of 14-31
 Shared data 14-5
 Shared memory
 adding segment with `onmode` 37-34
 allocating 14-26
 amount for sorting 14-29
 and blobpages 14-52
 and critical sections 14-46
 and multiple OnLine's 14-14
 and `SERVENUM` parameter 14-11
 and `SHMBASE` parameter 14-11
 attaching additional segments 14-12, 14-13
 attaching to 14-10
 base address 35-38
 buffer allocation 14-19
 buffer locks 14-31
 buffer pool 14-23, 20-8
 buffer table 14-18
 buffer, frequency of flushing 35-22
 changing residency with `onmode` 15-15, 37-30
 checkpoint 14-47
 chunk table 14-19
 communication 4-14
 communications portion 14-29
 connection 4-4, 4-6
 copying to a file 29-12
 created during initialization 9-6
 data-replication buffer 25-9
 dictionary cache 14-28
 dumps 35-17, 35-18
 during initialization 9-6
 effect of UNIX kernel parameters 15-3
 eliminating resource bottlenecks 30-11
 examining with `SMI` 36-4
 first segment 14-12
 for interprocess communication 14-5
 global pool 14-29
 header 14-13, 14-17
 heaps 14-28
 how much 14-10
 how utilities attach 14-11
 how virtual processors attach 14-11
 identifier 14-12
 initializing 9-3, 37-16
 initializing structures 9-7
 key value 14-11, 14-12
 largest allocation of resident portion 14-23
 latches 14-30
 locating a page 14-36
 lock table 14-20
 logical-log buffer 14-24
 lower boundary address problem 14-14
 message segments, mentioned 9-6
 mirror chunk table 14-19
 monitoring 29-13, 37-46
 mutexes 14-30
 OnLine requirements 14-11
 operating system segments 14-10
 page-cleaner table 14-21
 performance 14-5
 performance advantages 1-6
 physical-log buffer 14-25, 35-32
 pools 14-26
 portions 14-8
 purposes of 14-5
 re-initializing 15-14
 residency 15-15
 resident portion, flag 35-35
 resident segment, mentioned 9-6
 saving copy of with `onstat` 37-61
 segment identifier 14-12
 segments, dynamically added, size of 35-37
 session control block 14-27
 session data 14-27
 setting configuration parameters 15-3
 `SHMADD` parameter 14-26
 `SHMTOTAL` parameter 14-10
 `SHMVIRTSIZE` parameter 14-26
 size displayed by `onstat` 14-10, 37-48
 size of virtual portion 14-26
 sorting 14-28
 stacks 14-28
 `STACKSIZE` parameter 14-28
 stored procedures cache 14-29
 synchronizing buffer flushing 14-39
 tables 14-18
 `tblspace` table 14-21
 thread control block 14-27
 thread data 14-27
 thread isolation and buffer locks 14-31
 total size 14-13
 transaction table 14-22
 use of `SERVENUM` parameter 35-37

- user table 14-23
 - virtual portion 14-25, 14-26
 - virtual segments, mentioned 9-6
 - virtual segment, initial size 35-39
- Shared-memory buffer, maximum number 35-9
- Shared-memory connection
 - example 4-20
 - how a client attaches 14-10
 - in nettype field 4-14
 - in servicename field 4-16
 - message 12-27
 - message buffers 14-29
 - virtual processor 12-24
- SHM virtual processor 12-24
- SHMADD parameter
 - description of 14-26
 - specifying value 35-37
- SHMBASE parameter
 - attaching first shared-memory segment 14-12
 - description of 14-12, 35-38
 - mentioned 3-24
 - warning 14-13
- shmem file
 - and assertion failures 27-6
 - and DUMPSHMEM parameter 35-19
- shmem.xxx file 42-8
- shmkey
 - attaching additional segments 14-13
 - description of 14-12
- SHMTOTAL parameter
 - description of 14-10
 - specifying value 35-38
- SHMVIRTSIZE parameter
 - description of 35-39
 - specifying size of virtual shared memory 14-26
- Shutdown
 - graceful 8-5
 - immediate 8-5
 - mode, description of 7-4
 - taking off-line 8-6
- Shutdown script
 - multiple residency 6-8
 - steps to perform 3-29
- Single processor computer 12-17
- SINGLE_CPU_VP parameter
 - and single processor computer 12-17
 - description of 35-40
- Situations to avoid 28-3
- Sizing guidelines
 - logical log 18-6
 - physical log 20-5
- Sleep queues, description of 12-14
- Sleeping threads
 - forever 12-15
 - types of 12-14
- Slot table
 - description of 40-32
 - entry number 40-32
 - entry reflects changes in row size 40-33, 40-39
 - location on a dbspace page 40-30
 - relation to rowid 40-32
- SMI table
 - aborted table build 9-9
 - during initialization 9-9
 - monitoring buffer use 29-19
 - monitoring buffer-pool 29-22
 - monitoring checkpoints 29-12
 - monitoring chunks 29-50
 - monitoring data replication 29-60
 - monitoring databases 29-36
 - monitoring dbspaces 29-46
 - monitoring latches 29-23
 - monitoring locks 29-25
 - monitoring log buffer use 29-43
 - monitoring logical-log files 29-39
 - monitoring sessions 29-32
 - monitoring shared memory 29-15
 - monitoring virtual processors 29-29
 - preparation during initialization 9-9
- SOC virtual processors 12-24
- Sockets
 - connecting with 12-24
 - in nettype field 4-13
- Sorting
 - and shared memory 14-28
 - parallel 30-8
 - PSORT_NPROCS environment variable 30-8
- SPL statement recognized only by OnLine 31-15
- Split read 23-8
- SPX virtual processors 12-24
- SQL statement
 - ALTER INDEX 11-21

- branches recognized only by OnLine 31-14, 31-15
 - segments recognized only by OnLine 31-15
- SQLEXEC environment variable, example 4-27
- Sqlexec thread
 - and client application 12-10
 - as user thread 12-5
 - role in client/server connection 12-26
- sqlhosts file 3-8
 - and client redirection 25-22
 - dbservername field 4-12
 - defining multiple network addresses 12-29
 - description of 4-3, 4-10
 - entries for multiple interface cards 12-30
 - example 4-11
 - for initialization 3-26
 - in a learning environment 3-15
 - local loopback example 4-21
 - mentioned 1-4, 42-8
 - multiple connection types, example 4-18
 - multiple dbservernames 35-12
 - multiple residency 6-6
 - nettype field 4-12
 - network connection example 4-21
 - servicename field 4-15
 - shared-memory example 4-20
 - specifying network poll threads 12-25
 - syntax rules 4-12
- SQLRM environment variable 4-27
- SQLRMDIR environment variable 4-27
- Stack
 - and thread control block 12-13
 - description of 12-12
 - INFORMIXSTACKSIZE environment variable 14-28
 - monitoring stack size 29-30
 - pointer 12-13
 - size of 14-28
 - STACKSIZE parameter 14-28
 - thread 14-28
- STACKSIZE parameter
 - changing the stack size 14-28
 - description of 35-41
- STAGEBLOB parameter 35-41
- Standard database server 25-4
- Starting OnLine
 - and initializing disk space 3-27
 - in a learning environment 3-16
 - using oninit 37-16
- Startup script
 - multiple residency 6-8
 - multiple version of OnLine 3-29
- Statistics *See* onstat utility
- status_vset_volnum.itgr file 42-9
- Steps
 - for preparing a production environment 3-17
 - for using dbexport/dbimport 31-13
- Steps for preparing multiple residence 6-4
- Stored procedures cache 14-29
- stores6 demonstration database
 - copying Intro-10, 10
 - creating on INFORMIX-OnLine Intro-10, 10
 - overview Intro-9, 9
- Structured Query Language 1-4
- Structured query language. *See also* SQL statement.
- Switching between threads 12-13
- Symbolic link
 - using with TAPEDEV 35-42
- Symmetric multiprocessing, description of 12-3
- Synonym Name segment 31-15
- Synonym, moving 31-12
- sysfail.pidnum file 42-9
- Sysmaster database
 - description 36-3
 - functionality of 36-3
 - initialization 3-9
 - list of topics covered by 36-5
 - SMI tables 36-4
 - types of tables 36-3
 - warning 36-4
 - when created 36-4
- System catalog tables
 - and dictionary cache 14-28
 - disk space allocation for 40-63
 - how tracked 40-63
 - location of 10-20

- optimal storage of 10-36
- tracking a new database 40-63
- tracking a new table 40-65
- validating with oncheck 27-4
- System failure, defined 22-3
- System monitoring interface (SMI)
 - See also* SMI table
 - accessing SMI tables 36-5, 36-6
 - and locking 36-7
 - and SPL 36-6
 - and triggers 36-6
 - description 36-3
 - SMI tables map 36-21
 - tables
 - list of supported 36-7
 - sysadtinfo 36-8
 - sysaudit 36-9
 - syschkio 36-9
 - syschunks 36-10
 - sysdatabases 36-11
 - sysdbspaces 36-12
 - sysdri 36-13
 - sysextents 36-13
 - syslocks 36-13
 - syslogs 36-14
 - sysprofile 36-15
 - sysptprof 36-16
 - sysesprof 36-17
 - syseswts 36-20
 - sysstabnames 36-20
 - sysvpprof 36-21
 - using to monitor OnLine 29-8
 - using to obtain onstat information 36-24
 - viewing tables with dbaccess 36-5
- System startup script, multiple residency 6-8
- System timer 12-30

T

Table

- creating, what happens on disk 40-63, 40-64
- description of 10-21
- high-use 10-34
- identifying its dbspace 40-19
- isolating high-access 10-31
- management of 11-21
- migration
 - See* Migration

- monitoring with SMI 36-20
- moving, using onunload/onload 31-8
- placing in a specific dbspace 10-22
- pseudo- tables 36-4
- purpose of 10-21
- recommendations for storage 10-33
- relationship to extent 10-21
- SMI tables 36-4
- spreading across disks 10-33
- storage on middle partition of disk 10-34
- temporary 10-23
 - cleanup during shared-memory initialization 10-24
 - effects of creating 40-67
 - message reporting cleanup 38-12
 - storage of explicit 10-24
 - storage of implicit 10-25

Table Name segment 31-15

Tape device

- block size 35-23
- in learning environment 3-14

Tape management 3-30

TAPEBLK parameter

- description of 35-41
- mentioned 3-23

TAPEDEV parameter

- description of 35-42
- in a learning environment 3-14
- in a production environment 3-19
- mentioned 3-23, 3-26
- using a symbolic link 35-42
- with onunload/onload 31-7

TAPESIZE parameter

- description of 35-44
- mentioned 3-23

TBCONFIG environment variable 42-7

Tbldspace

- description of 10-25
- displaying information with onstat 37-65
- identifying its dbspace 40-19
- maximum open 35-44
- monitoring active tblspaces 29-26
- monitoring with SMI 36-16
- number 40-19
- number displayed 37-66
- number elements 40-19
- purpose of 10-25

-
- temporary tblspace during
 - initialization 9-8
 - types of pages contained in 10-25
 - Tblspace number
 - components of 40-19
 - description of 40-19
 - displaying with onstat -t 37-66
 - includes dbspace number 40-19
 - retrieving it from systables 40-19
 - Tblspace table
 - contents of 14-21
 - description of 14-21
 - hash table 14-22
 - Tblspace tblspace
 - bit-map page 40-20
 - description of 40-17
 - location in a chunk 40-13
 - location in root dbspace 40-4
 - size 40-20
 - structure and function 40-17
 - tracking new tables 40-65
 - TBLSPACES parameter
 - description of 35-44
 - mentioned 3-24
 - purpose of 14-22
 - tuning for performance 30-18
 - TCP/IP communication protocol
 - in hostname field 4-15
 - in nettype field 4-14
 - in servicename field 4-16
 - support of 1-8
 - using 4-8
 - tctermcap archive attributes file 42-9
 - Template file for configuration 3-8
 - Template for ONCONFIG file 42-7
 - Temporary dbspace
 - advantages of 10-18
 - and data replication 25-31, 26-6
 - and DBSPACETEMP 10-18
 - and performance 10-19
 - described 10-18
 - Temporary disk space
 - operations requiring 10-23
 - recommendations for using 30-7
 - Temporary table
 - DBSPACETEMP parameter 35-13
 - description of 10-23
 - during initialization 9-8
 - explicit 10-23
 - implicit 10-23
 - rules for use 35-13
 - where stored 10-24
 - with oninit utility 37-17
 - TERM environment variable 3-9
 - TERMCAP environment variable 3-9
 - TERMINFO environment variable 3-9
 - TEXT data type
 - Dirty Read isolation 14-51
 - migrating from INFORMIX-SE 31-16
 - migrating to INFORMIX-SE 31-14
 - requires 4-bit bit map 40-23, 40-25
 - storage on disk 10-16
 - Text editor
 - setting performance configuration parameters 15-13
 - setting shared memory parameters 15-8, 15-11
 - setting virtual processor parameters 13-5
 - Thread
 - accessing shared buffers 14-32
 - and heaps 14-28
 - and stacks 14-28
 - concurrency control 14-30
 - context of 12-10
 - control block 12-10, 14-27
 - description of 12-5
 - for client applications 12-4
 - for primary session 12-10
 - for recovery 12-5
 - how virtual processors service 12-9
 - internal 12-5, 12-16
 - kernel asynchronous I/O 12-22
 - migrating 12-14
 - mirroring 12-5
 - monitoring 29-29
 - multiple concurrent 12-10
 - ON-Monitor 12-5
 - onstat information 37-69
 - page cleaner 12-5, 14-39
 - relationship to a process 12-5
 - scheduling and synchronizing 12-10
 - session 12-5, 12-16
 - sleeping 12-15, 14-38
 - supporting data replication 25-13
 - switching between 12-13
 - user 12-5
 - waking up 12-14
 - yielding 12-10

- Thread control block 14-27
- Time-out condition 37-54
- Timestamp
 - blob pair 14-50
 - blob timestamps on a blobpage 40-61
 - description of 14-50
 - location
 - blobpage 40-60, **40-61**
 - dbspace blob page 40-57, **40-57**
 - dbspace page 40-30
 - page-header and page-ending pair 14-50, 40-31
 - role in
 - data consistency 14-50
 - flushing physical-log buffer 14-42
 - synchronizing buffer flushing 14-41
- TLI. *See* Transport-level interface
- Transaction
 - factors which prevent closure 18-14
 - global
 - definition of 32-5
 - determining if implemented consistently 33-4
 - identification number, GTRID 33-7
 - tracking 32-27
 - kill with onmode -Z 37-32
 - maximum open 35-45
 - monitoring 29-33
 - pending 37-68
 - piece of work, definition of 32-5
 - two-phase commit, examples 32-7
- Transaction logging. *See* Logging
- Transaction table
 - description of 14-22
 - tracking with onstat 14-22
- TRANSACTIONS parameter
 - description of 35-45
 - mentioned 3-24
- Transport-level interface
 - connecting with 12-24
 - in nettype field 4-13
 - virtual processors 12-24
- Tuning
 - large number of users 35-29
 - use of NETTYPE parameter 35-28
- Two-phase commit protocol
 - automatic recovery 32-9
 - administrator's role 32-10
 - mechanisms for coordinator recovery 32-10
 - mechanisms for participant recovery 32-14
 - race condition 32-17
- configuration parameters for 32-34
- coordinator definition 32-5
- coordinator recovery mechanism 32-11
- description of 32-3, 32-5
- errors messages for 32-27
- global transaction definition 32-5
- global transaction identification number 33-7
- heuristic decisions
 - heuristic end-transaction 32-25
 - heuristic rollback 32-21
 - types of 32-20
- independent action
 - definition of 32-18
 - resulting in error condition 32-19
 - resulting in heuristic decisions 32-20
 - results of 32-19
 - what initiates 32-18
- logical-log records for 32-28
- messages 32-6
- participant recovery 32-14
- participant, definition of 32-5
- piece of work, definition of 32-5
- post-decision phase 32-6, 32-9
- precommit phase 32-6
- presumed-abort optimization 32-10, 32-17
- requirements for flushing logical log records 32-29
- role of current server 32-5
- use of DEADLOCK_TIMEOUT 32-34
- use of TXTIMEOUT 32-34

- TXTIMEOUT parameter
 - and onmode -Z 32-26
 - description of 32-34, 35-45
 - in two-phase commit protocol 32-34
 - mentioned 3-25
 - role in automatic recovery 32-11, 32-14, 32-17
- Types of buffer writes 14-42

U

- Updates
 - disk 3-21

- Unbuffered transaction logging.
 - See* Logging
 - unique id 18-8
 - Units of storage 10-3
 - UNIX devices
 - creating a link to a pathname 11-6
 - ownership, permissions on character-special 11-5
 - when are offsets needed 11-5
 - UNIX files
 - ownership, permissions on cooked files 11-4
 - using for data storage 10-8
 - UNIX kernel parameters
 - description of 15-3
 - initial values 3-27
 - lower boundary address parameter 14-14
 - UNIX link command 11-6
 - UNIX shutdown script 3-29
 - UNLOAD statement
 - and NLS 31-10
 - using 31-10, 31-11
 - with dbload utility 31-12
 - Update
 - database performance 30-7
 - effects of mass updates on performance 30-11
 - Upgrading OnLine from an earlier version 3-7
 - USEOSTIME parameter
 - affect on performance 30-20
 - description of 30-20, 35-46
 - User guidelines to improve performance 30-10
 - User session
 - monitoring 29-29
 - monitoring with SMI 36-18
 - status codes 37-66
 - User table
 - description of 14-23
 - maximum number of entries 14-23
 - User thread
 - acquiring a buffer 14-36
 - description of 12-5
 - in critical sections 14-46
 - monitoring 14-23
 - tracking 14-23
 - User-specified ID and passwd 4-9
 - Users, number of, in NETTYPE parameter 35-28
 - USERTHREADS parameter
 - and communications portion (shared memory) 14-30
 - and shared-memory buffers 14-19
 - mentioned 3-24
 - tuning for performance 30-18
 - use by ON_RECVRY_THREADS 35-32
 - use in NETTYPE parameter 35-28
 - used by OFF_RECVRY_THREADS parameter 35-31
 - Utilities
 - attaching to shared-memory 14-11
 - gcore 35-17, 35-18
 - oncheck 37-6
 - oninit 37-16
 - onload 37-18
 - onlog 37-23
 - onmode 37-27
 - onparams 37-37
 - onspaces 37-40
 - onstat 37-46
 - ontape 37-70
 - onunload 37-77
- ## V
- VARCHAR data type
 - byte locks 14-21
 - changing locale 31-17
 - implications for data row storage 40-35
 - indexing considerations 40-54
 - requires 4-bit bit map 40-23, 40-25
 - storage considerations 40-33
 - Version
 - connecting to different 4-26
 - Version 5.0 Relay Module. *See* Relay Module, version 5.0
 - Version 6.0 Relay Module. *See* Relay Module, version 6.0
 - View Name segment 31-15
 - View, moving 31-12
 - Virtual portion (shared memory)
 - adding a segment 15-16
 - contents of 14-25, 14-26
 - global pool 14-29

- mentioned 9-6
- setting configuration parameters 15-9
- size of 14-26
- stacks 14-28
- stored procedures cache 14-29

Virtual processor

- adding and dropping 12-9
- add/remove with onmode 37-34
- ADM class 12-14, 12-30
- ADT class 12-31
- advantages 12-6
- AIO class 12-23
- AIO, how many 12-23
- as multithreaded process 12-5
- attaching to shared-memory 14-11
- binding to CPUs 12-9
- classes of 12-5, 12-16
- CPU class 12-16
- description of 12-4
- disk I/O 12-20
- dropping (CPU) in on-line mode 13-9
- during initialization 9-7
- how threads serviced 12-9
- LIO class 12-21
- LIO, how many 12-21
- logical-log I/O 12-21
- memory and resources 12-7
- monitoring 29-27
- moving a thread 12-7
- network 12-24, 12-25
- number in AIO class 35-30
- number in CPU class 35-31
- OPT (optical) class 12-30
- parallel processing 12-8
- physical log I/O 12-22
- PIO class 12-21
- PIO, how many 12-22
- priority aging 35-30
- setting configuration parameters 13-3
- sharing processing 12-7

VP class in NETTYPE parameter 12-25, 35-28

VP.servername.xx file 42-9

W

Wait queue

- and buffer locks 14-31
- description of 12-15

Waking up threads 12-14

Warning

- buildsmi script 36-4
- files on NIS systems 4-9
- interpreting after running oncheck -cc 27-5

Whitespace in ONCONFIG file 35-6

WORM devices 1-8

Write types

- chunk write 14-43
- efficiency trade-off between LRU and chunk writes 30-14
- foreground write 14-43
- LRU write 14-43

Y

Yielding threads

- at predetermined point 12-11
- description of 12-10
- on some condition 12-11

ypcat hosts, UNIX command 4-9

ypcat services, UNIX command 4-9

Symbols

\$INFORMIXDIR/etc/sqlhosts. *See* sqlhosts file

.informix file

- mentioned 3-30, 42-5
- multiple residency 6-8

.infos.dbservername file

- regenerate 37-36

.infos.dbservername file

- description of 42-6
- regenerate 37-36

.inf.servicename file 42-6

.netrc file 4-9

.rhosts file 4-9

/dev/null

- in learning environment 3-14

/etc/hosts file

- and client redirection 25-23
- multiple residency 6-6

/etc/hosts.equiv file 4-9, 35-43

/etc/passwd file

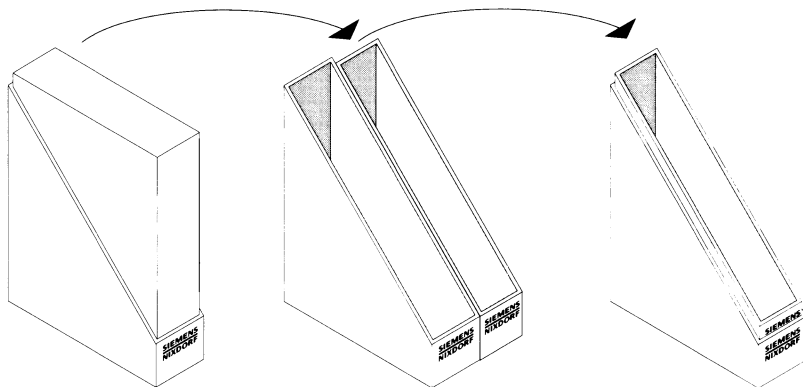
- mentioned 4-9

/etc/services file
and client redirection 25-23
multiple residency 6-6
/etc/shadow file
mentioned 4-9
/.rhosts file 35-43

Sammelboxen / Organizer cases

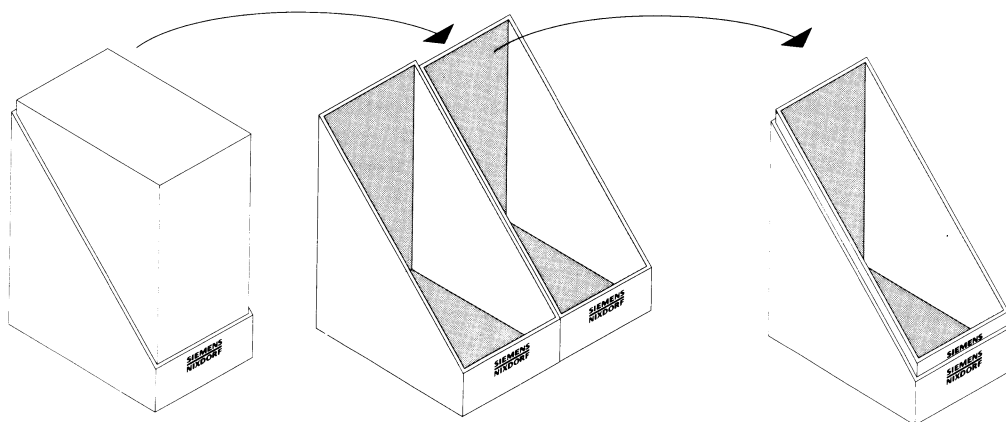
Für Handbücher des vorliegenden Formates bieten wir zweiteilige Sammelboxen in zwei unterschiedlichen Größen an. Der Bestellvorgang entspricht dem für Handbücher.

Two-part organizer cases are available in two sizes for storing manuals in the present format. The ordering procedure is the same as for manuals.



Breite: ca. 5 cm
Bestellnummer: U3775-J-Z18-1

Width: approx. 5 cm
Order No.: U3775-J-Z18-1



Breite: ca. 10 cm
Bestellnummer: U3776-J-Z18-1

Width: approx. 10 cm
Order No.: U3776-J-Z18-1

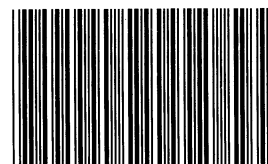
943796

Herausgegeben von / Published by
Siemens Nixdorf Informationssysteme AG
D-33094 Paderborn
D-81730 München

Bestell-Nr./Order No. **U9636-J-Z265-2-7600**
Printed in the Federal Republic of Germany
4610 AG 04941.8 (7660) F

Das Papier dieser Broschüre erfüllt
unsere Forderungen nach einem umwelt-
freundlichen Papier. Das Rohpapier wird
aus chlorfrei gebleichtem Zellstoff
hergestellt.

This brochure is printed on environmen-
tally friendly paper, cellulose treated
with chlorine-free bleach.



9Y505830