

Progress Database Administration Guide and Reference

© 2002 Progress Software Corporation. All rights reserved.

Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. This manual is also copyrighted and all rights are reserved. This manual may not, in whole or in part, be copied, photocopied, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from Progress Software Corporation.

The information in this manual is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear in this document.

The references in this manual to specific platforms supported are subject to change.

Progress, Powered by Progress, Progress Fast Track, Progress Profiles, Partners in Progress, Partners en Progress, Progress en Partners, Progress in Progress, P.I.P., Progress Results, ProVision, ProCare, ProtoSpeed, SmartBeans, SpeedScript, and WebSpeed are registered trademarks of Progress Software Corporation in the U.S. and other countries. A Data Center of Your Very Own, Allegrix, Apptivity, AppsAlive, AppServer, ASPen, ASP-in-a-Box, Empowerment Center, Fathom, Future Proof, IntelliStream, OpenEdge, PeerDirect, POSSE, POSSENET, Progress Dynamics, Progress Software Developers Network, SectorAlliance, SmartObjects and WebClient are trademarks or service marks of Progress Software Corporation in the U.S. and other countries.

SonicMQ is a registered trademark of Sonic Software Corporation in the U.S. and other countries.

Vermont Views is a registered trademark of Vermont Creative Software in the U.S. and other countries.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Any other trademarks and/or service marks contained herein are the property of their respective owners.

Progress Software Corporation acknowledges the use of Raster Imaging Technology copyrighted by Snowbound Software 1993-1997, the IBM XML Parser for Java Edition, and software developed by the Apache Software Foundation (http://www.apache.org/).

© IBM Corporation 1998-1999. All rights reserved. U.S. Government Users Restricted Rights — Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Progress is a registered trademark of Progress Software Corporation and is used by IBM Corporation in the mark Progress/400 under license. Progress/400 AND 400® are trademarks of IBM Corporation and are used by Progress Software Corporation under license.

The implementation of the MD5 Message-Digest Algorithm used by Progress Software Corporation in certain of its products is derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm.

May 2002



Product Code: 4590
Item Number: 89434;V9.1D

Contents

Preface . . . . . **xxiii**
Purpose xxiii
Audience xxiii
Organization of This Manual xxiii
Part I Planning xxiii
Part II Administration xxiv
Part III Reference xxv
How To Use This Manual xxvi
Typographical Conventions xxvi
Syntax Notation xxvii
Example Procedures xxxi
Progress Messages xxxiv
Other Useful Documentation xxxvi
Getting Started xxxvi
Development Tools xxxvii
Reporting Tools xxxviii
4GL xxxix
Database xl
DataServers xl
SQL-89/Open Access xli
SQL-92 xli
Deployment xlii
WebSpeed xlii
Reference xliii

1. The Progress Database **1–1**
1.1 Progress Database Architecture 1–2
1.1.1 Other Database-related Files 1–4
1.2 Storage Design Overview 1–5

1.2.1 Storage Objects 1–7
1.2.2 Extents 1–7
1.2.3 Storage Areas 1–7
1.3 Determining Configuration Variables 1–9
1.3.1 System Platform 1–9
1.3.2 Connection Modes 1–9
1.3.3 Database Location 1–10
1.3.4 Database Configurations. 1–10
1.4 Operating System Resources 1–12
1.5 Multi-threaded Architecture 1–13
1.6 Multi-tier and Cluster Configurations 1–14
1.7 Self-service and Network Clients 1–15
1.8 Relative- and Absolute-path Databases 1–15

2. Administrative Planning 2–1
2.1 Block Size Considerations 2–2
2.1.1 Guidelines For Choosing Storage Area Locations 2–3
2.2 Calculating Database Disk Requirements 2–3
2.3 Storing Database Extents On Raw Partitions 2–7

3. Progress Database Limits 3–1
3.1 Database Block Sizes 3–2
3.2 Number and Size Of Storage Areas 3–2
3.3 Maximum Number Of Records Per Block 3–5
3.4 Table and Index Limits 3–5
3.5 Number Of Sequences 3–5
3.6 Maximum Size Of the Primary Recovery (BI) Area 3–6
3.7 Maximum Database Size 3–6
3.8 Number Of Connections Per Database 3–7
3.9 Number Of Simultaneous Transactions Per Database 3–7
3.10 Size Of Database Buffer Pool 3–7
3.11 Progress Database Name Limits 3–8
3.12 Applicable Operating System Limits 3–8
3.13 Data Types and Values 3–10

4. Creating and Deleting Databases 4–1
4.1 Ways To Create a Progress Database 4–2
4.2 Creating a Database With PROSTRCT CREATE 4–2
4.2.1 Creating a Structure Description File 4–3
4.2.2 Create a Database Structure Extent 4–10
4.2.3 Adding Schema To a Void Database 4–11
4.3 Creating a Database With the PRODB Utility 4–12
4.4 Creating a Database With the Data Administration Tool 4–14
4.5 Migrating Version 8 Databases To Version 9 Databases 4–15

4.5.1 Converting a Single-volume Version 8 Database With PROCOPY 4–16
4.5.2 Converting a Single-volume Version 8 Database With PROREST 4–17
4.5.3 Converting a Single-volume Version 8 Database With PROSTRCT 4–18
4.5.4 Converting a Version 8 Database To Version 9. 4–18
4.6 Using the Schema Mover After Conversion 4–19
4.7 Copying a Database 4–22
4.8 AutoConvert Utility 4–24
4.9 Deleting a Database 4–24

5. Starting Up and Shutting Down 5–1
5.1 The Progress Explorer Framework 5–2
5.1.1 AdminServer 5–2
5.1.2 Progress Explorer 5–2
5.1.3 Command-line Configuration Utilities 5–3
5.1.4 Using the DBMAN Command-Line Utility 5–4
5.2 Starting a Server Or Broker 5–5
5.2.1 Using the PROSERVE Command 5–5
5.2.2 Specifying International Character Sets 5–6
5.2.3 Network Addressing With Progress (- S and -H) 5–7
5.2.4 Starting Network Brokers and Servers. 5–7
5.2.5 Starting Multiple Brokers Using the Same Protocol 5–8
5.2.6 Accessing a Server Behind a Firewall 5–9
5.3 Starting and Stopping Background Writers 5–10
5.3.1 Starting and Stopping an APW 5–10
5.3.2 Starting and Stopping a BIW 5–11
5.3.3 Starting and Stopping an AIW 5–11
5.4 Stopping a Server Or Broker 5–12
5.4.1 PROSHUT Command 5–12
5.4.2 PROMON Shut Down Database Option 5–15

6. Backup and Recovery Strategies 6–1
6.1 Identifying Files For Back Up 6–2
6.2 Determining the Type Of Backup 6–3
6.2.1 Incremental Backups 6–4
6.2.2 Online Backups 6–5
6.2.3 Offline Backups 6–6
6.3 Choosing Backup Media 6–6
6.4 Creating a Backup Schedule 6–7
6.4.1 Database Integrity 6–7
6.4.2 Database Size 6–8
6.4.3 Time 6–8

6.4.4 Unscheduled Backups 6–9
6.5 Developing a Recovery Plan 6–9
6.5.1 Lost Transactions 6–10
6.5.2 Duration Of Offline Applications 6–10
6.5.3 Time Needed For Recovery 6–11
6.5.4 Distributed Transactions Occurring Inconsistently 6–12
6.6 Recovery Guidelines 6–13

7. Backing Up a Database. 7–1
7.1 Performing a Progress Backup 7–2
7.1.1 Performing an Online Full Backup With PROBKUP 7–3
7.1.2 Progress Backup Options 7–5
7.1.3 Testing Backups 7–6
7.1.4 Archiving Backups 7–8
7.2 Performing an Offline Progress Backup 7–9
7.3 Performing an Online Progress Backup 7–10
7.4 Using Database Quiet Points 7–11
7.5 Performing a Non-Progress Backup 7–13
7.6 Database Backup Examples 7–14
7.6.1 UNIX Incremental Backup Example 7–14
7.6.2 Windows Full Backup Example. 7–17
7.6.3 Incremental Backup Example 7–20
7.7 Verifying a Backup 7–23
7.8 CRC Codes and Redundancy In Backup Recovery 7–24
7.8.1 CRC Codes. 7–24
7.8.2 Error-correction Blocks 7–25
7.9 Restoring a Database 7–25
7.9.1 Using the PROREST Utility To Restore a Database 7–26
7.9.2 Important Rules For Restoring Backups 7–27
7.9.3 Obtaining Storage Area Descriptions Using PROREST 7–28
7.9.4 Database Restore Examples 7–29

8. Recovering a Database. 8–1
8.1 Introduction To Recovery Mechanisms 8–2
8.1.1 Crash Recovery 8–3
8.1.2 Roll-forward Recovery. 8–4
8.1.3 Two-phase Commit 8–6
8.2 File Locations That Ensure Safe Recovery 8–6
8.3 Sample Recovery Plans 8–7
8.3.1 Example 1: Low Availability Requirements 8–7
8.3.2 Example 2: Moderate Availability Requirements 8–8
8.3.3 Example 3: Moderate-to-High Availability Requirements 8–9
8.3.4 Example 4: High Availability Requirements 8–10
8.3.5 Sample Recovery Scenarios. 8–11

8.4 After-imaging and Roll-forward Recovery Commands 8–16
8.5 Recovering From System Failures 8–17
8.5.1 System Crash While Running RFUTIL ROLL FORWARD. . . . 8–17
8.5.2 System Crash While Running Other Utilities 8–17
8.5.3 System Crash While Backing Up the Database. 8–18
8.5.4 System Crash While Database Is Up. 8–18
8.6 Recovering From Media Failures 8–18
8.6.1 Loss Of the DB Files, BI Files, Or Both 8–19
8.6.2 Loss Of the AI File 8–19
8.6.3 Loss Of Database Backup 8–20
8.6.4 Loss Of Transaction Log File 8–21
8.7 Recovering From a Full Disk 8–21
8.7.1 After-image Area Disk 8–22
8.7.2 Control Or Primary Recovery Area Disk 8–22
8.7.3 Transaction Log File Disk 8–23
8.8 Truncating the BI File 8–24
8.9 Recovering From a Crash 8–24
8.10 Recovering From a Lost Or Damaged Control Area 8–25
8.11 Unlocking Damaged Databases 8–26
8.12 Dumping Tables From a Damaged Database 8–27
8.13 Forcing Access To a Damaged Database With the -F Parameter 8–28

9. Maintaining Database Structure 9–1
9.1 The Progress Structure Utility 9–2
9.2 Progress Structure Statistics Utility 9–2
9.3 Progress Structure List Utility 9–3
9.4 Progress Structure Add Utility 9–5
9.4.1 PROSTRCT ADD and Pathnames 9–6
9.4.2 Adding Extents to Existing Storage Areas 9–7
9.5 Progress Structure Remove Utility 9–8
9.6 Maintaining Indexes and Tables 9–9
9.6.1 Moving Tables 9–9
9.6.2 Moving Indexes 9–11
9.6.3 Compacting Indexes 9–11
9.7 Using Virtual System Tables To Obtain Status Of Administration Utilities 9–12

10. Maintaining Security 10–1
10.1 Establishing a Progress User ID and Password 10–2
10.1.1 Progress User ID 10–2
10.1.2 Progress Password 10–2
10.1.3 Validating a Progress User ID and Password 10–3
10.2 Establishing Authentication For Your Progress Database 10–3
10.2.1 4GL Tables Only 10–3
10.2.2 SQL-92 Tables Only 10–3

10.2.3 Both 4GL and SQL-92 Tables 10–4
10.3 Connection Security 10–5
10.3.1 Designating a Security Administrator 10–5
10.3.2 Adding a User 10–7
10.3.3 Deleting a User 10–9
10.3.4 Changing a Password 10–9
10.4 Running a User Report 10–10
10.5 Schema Security 10–11
10.5.1 Freezing and Unfreezing a Table 10–12
10.6 Operating Systems and Database Security 10–12

11. After-Imaging 11–1
11.1 After-image Areas and Extents 11–2
11.2 Estimating After-imaging Disk Space Requirements 11–3
11.3 Creating After-image Areas 11–4
11.4 Enabling After-imaging 11–5
11.5 Managing After-imaging Files 11–6
11.5.1 Monitoring AI File Status 11–6
11.5.2 Switching To a New AI File 11–7
11.5.3 Archiving an AI File 11–8
11.6 Performing Roll-forward Recovery 11–12
11.7 Disabling After-imaging 11–13

12. Using Two-phase Commit 12–1
12.1 Distributed Transactions 12–2
12.2 How The Database Engine Implements Two-phase Commit 12–3
12.2.1 Two-phase Commit and Roll-forward Recovery 12–6
12.3 Enabling Two-phase Commit 12–7
12.3.1 Modifying the Database Nickname and Priority 12–8
12.3.2 Transaction Log Area 12–9
12.4 Detecting Limbo Transactions 12–10
12.5 Resolving Limbo Transactions 12–11
12.5.1 Resolving Limbo Transactions With the Database In Use 12–11
12.5.2 Resolving Limbo Transactions With the Database Shut Down . 12–15
12.5.3 Resolving Limbo Transaction Scenarios 12–17
12.6 Deactivating Two-phase Commit 12–19
12.7 Case Study 12–19

13. Dumping and Loading 13–1
13.1 Overview Of Dumping and Loading 13–2
13.1.1 Dump and Load Limitations 13–3
13.2 Dumping 4GL Database Definitions 13–3
13.2.1 Definition File Format 13–4
13.2.2 Dumping 4GL Definitions 13–6

13.2.3 Creating an Incremental 4GL Data Definitions File 13–7
13.2.4 Dumping Sequence Definitions 13–8
13.2.5 Dumping Auto-connect Records 13–9
13.3 Dumping Database Contents 13–9
13.3.1 Dumping Table Contents With PROUTIL. 13–10
13.3.2 Dumping Field Contents With PROUTIL 13–12
13.3.3 Dumping Table Contents With a Data Tool 13–14
13.3.4 Contents File Format 13–15
13.3.5 Dumping Sequence Values 13–17
13.3.6 Dumping User Table Contents. 13–17
13.3.7 Dumping an SQL View File's Contents 13–18
13.4 Loading Database Definitions 13–18
13.4.1 Loading Table Definitions 13–18
13.4.2 Loading Updated 4GL Data Definitions 13–19
13.5 Loading Database Contents 13–20
13.5.1 Loading Table Contents With the PROUTIL Command. 13–21
13.5.2 Loading Table Contents With a Data Tool 13–22
13.5.3 Loading User Table Contents 13–23
13.5.4 Loading an SQL View File Contents 13–24
13.5.5 Loading Sequence Values 13–24
13.6 Bulk Loading 13–25
13.6.1 Creating a Bulk Loader Description File. 13–25
13.6.2 Modifying a Bulk Loader Description File 13–26
13.6.3 Loading Table Contents With the Bulk Loader Qualifier 13–27
13.7 Reconstructing Bad Load Records 13–30
13.8 Specialized Dump and Load Techniques 13–30
13.8.1 Creating a Starting Version Of a Database 13–30
13.8.2 Using a Constant Table In Multiple Databases 13–32
13.8.3 Economizing Disk Space 13–32
13.8.4 Optimizing Data For Sequential Access. 13–33
13.8.5 Optimizing Data For Random Access 13–34
13.8.6 Using No-integrity Mode Bulk Data Loading 13–36
13.8.7 Using 4GL Tools To Save Space By Deactivating Indexes . . . 13–36

14. Managing Performance 14–1
14.1 Introduction To Performance Managing 14–2
14.2 Tools For Monitoring Performance 14–3
14.2.1 PROMON Utility 14–3
14.2.2 Virtual System Tables 14–3
14.2.3 NT Performance Monitor 14–3
14.3 Server Performance Factors 14–4
14.4 CPU Utilization 14–5
14.4.1 Idle CPU. 14–5
14.4.2 Unproductive CPU Processing. 14–5

14.5 Disk I/O 14–6
14.5.1 Database I/O 14–6
14.5.2 Before-image I/O 14–18
14.5.3 After-image I/O 14–27
14.5.4 Direct I/O 14–30
14.6 Memory Utilization 14–31
14.6.1 Monitoring Memory Utilization 14–31
14.6.2 Controlling Memory Use 14–32
14.7 Operating System Resources 14–33
14.7.1 Processes 14–33
14.7.2 Semaphores 14–34
14.7.3 Spin Locks 14–37
14.7.4 File Descriptors 14–37
14.8 Database Fragmentation 14–37
14.8.1 Analyzing Database Fragmentation 14–37
14.8.2 Eliminating Database Fragmentation 14–39
14.9 Index Use 14–39
14.9.1 Analyzing Index Use 14–40
14.9.2 Compacting Indexes 14–41
14.9.3 Rebuilding Indexes 14–42
14.10 Virtual System Tables 14–47

15. Replicating Data 15–1
15.1 Replication Schemes 15–2
15.1.1 Trigger-based Replication 15–2
15.1.2 Log-based Site Replication 15–2
15.2 Replication Models 15–2
15.2.1 Synchronous Model 15–2
15.2.2 Asynchronous Model 15–3
15.2.3 Database Ownership Models 15–3
15.3 Implementing Log-based Site Replication 15–7
15.3.1 Log-based Replication Procedure 15–7

16. Using the Event Log 16–1
16.1 Progress Version 9 Event Log File 16–2
16.2 Managing the Event Log File Size 16–2
16.3 Event Logging On Windows 16–2
16.3.1 Managing Progress Events On Windows 16–3
16.3.2 Understanding the Event Log Components 16–4
16.3.3 The Event Log and the Registry 16–7

17. Startup and Shutdown Commands 17–1
17.1 Startup Command Syntax 17–2
17.2 Database Startup and Shutdown Commands 17–3

PROAIW Command 17–4
PROAPW Command 17–5
PROBIW Command 17–6
PROQUIET Command 17–7
PROSERVE Command 17–9
PROSHUT Command 17–11
PROWDOG Command 17–15

18. Database Startup Parameters 18–1
18.1 Issuing Startup Parameters 18–2
18.2 Database Server Performance Parameters 18–2
18.3 Database Server-type Parameters 18–4
18.4 Database Server Internationalization Parameters 18–4
18.5 Database Server Statistics Parameters 18–5
18.6 Database Server Network Parameters 18–6
18.7 Startup Parameter Usage Categories 18–8
18.8 Alphabetical Listing Of Database Startup Parameters 18–8
18.8.1 AdminServer Port (-adminport) 18–8
18.8.2 After-image Buffers (-aibufs) 18–9
18.8.3 After-image Stall (-aistall) 18–10
18.8.4 Blocks In Database Buffers (-B) 18–10
18.8.5 Base Index (-baseindex) 18–11
18.8.6 Base Table (-basetable) 18–12
18.8.7 Before-image Buffers (-bibufs) 18–13
18.8.8 Threshold Stall (-bistall) 18–13
18.8.9 Recovery Log Threshold (-bithold) 18–14
18.8.10 SQL-92 Server Java Classpath (-classpath) 18–14
18.8.11 Conversion Map (-convmap) 18–15
18.8.12 Case Table (-cpcase) 18–15
18.8.13 Collation Table (-cpcoll) 18–16
18.8.14 Internal Code Page (-cpinternal) 18–17
18.8.15 Log File Code Page (-cplog) 18–18
18.8.16 Print Code Page (-cpprint) 18–18
18.8.17 R-code In Code Page (-cprcodein) 18–19
18.8.18 Stream Code Page (-cpstream) 18–20
18.8.19 Terminal Code Page (-cpterm) 18–21
18.8.20 Direct I/O (-directio) 18–21
18.8.21 Event Level (-evtlevel) 18–22
18.8.22 Before-image Cluster Age (-G) 18–23
18.8.23 Group Delay (-groupdelay) 18–23
18.8.24 Host Name (-H) 18–24
18.8.25 Hash Table Entries (-hash) 18–24
18.8.26 No Crash Protection (-i) 18–25
18.8.27 Index Range Size (-indexrangesize) 18–26

18.8.28 Lock Table Entries (-L) 18–26
18.8.29 Auto Server (-m1) 18–27
18.8.30 Manual Server (-m2) 18–28
18.8.31 Secondary Login Broker (-m3) 18–28
18.8.32 Maximum Clients Per Server (-Ma) 18–29
18.8.33 Maximum Dynamic Server (-maxport) 18–29
18.8.34 Delayed BI File Write (-Mf) 18–30
18.8.35 Minimum Clients Per Server (-Mi) 18–31
18.8.36 Minimum Dynamic Server (-minport) 18–32
18.8.37 Maximum Servers (-Mn) 18–32
18.8.38 Servers Per Protocol (-Mp) 18–33
18.8.39 Maximum Servers Per Broker (-Mpb) 18–33
18.8.40 VLM Page Table Entry Optimization (-Mpte) 18–34
18.8.41 Shared-memory Overflow Size (-Mxs) 18–34
18.8.42 Network Type (-N) 18–35
18.8.43 Number Of Users (-n) 18–35
18.8.44 Parameter File (-pf) 18–36
18.8.45 Pin Shared Memory (-pinshm) 18–36
18.8.46 Configuration Properties File (-properties) 18–37
18.8.47 Buffered I/O (-r) 18–37
18.8.48 Service Name (-S) 18–38
18.8.49 Semaphore Sets (-semsets) 18–39
18.8.50 Server Group (-servergroup) 18–40
18.8.51 Spin Lock Retries (-spin) 18–40
18.8.52 Table Range Size (-tablerangesize) 18–41

19. Database Administration Utilities 19–1
DBMAN Utility 19–3
_DBUTIL CMPDB Qualifier 19–7
PROADSV Utility 19–8
PROBKUP Utility 19–10
ProControl Utility 19–14
PROCOPY Utility 19–17
PRODB Utility 19–19
PRODEL Utility 19–22
Progress Explorer Utility 19–23
PROLOG Utility 19–24
PROMON Utility 19–25
PROMON User Control Option 19–27
PROMON Locking and Waiting Statistics Option 19–30
PROMON Block Access Option 19–32
PROMON Record Locking Table Option 19–35
PROMON Activity Option 19–37
PROMON Shared Resources Option 19–41

PROMON Database Status Option 19–43
PROMON Shut Down Database Option 19–45
PROMON Transactions Control Option 19–46
PROMON Resolve Limbo Transactions Option 19–48
PROMON Coordinator Information Option 19–49
PROMON Modify Defaults Option 19–50
PROREST Utility 19–52
PROSTRCT Utility 19–54
PROSTRCT ADD Qualifier 19–56
PROSTRCT BUILDDB Qualifier 19–57
PROSTRCT CREATE Qualifier 19–58
PROSTRCT LIST Qualifier 19–60
PROSTRCT REMOVE Qualifier 19–61
PROSTRCT REPAIR Qualifier 19–63
PROSTRCT STATISTICS Qualifier 19–64
PROSTRCT UNLOCK Qualifier 19–66
PROUTIL Utility 19–67
PROUTIL 2PHASE BEGIN Qualifier 19–71
PROUTIL 2PHASE COMMIT Qualifier 19–73
PROUTIL 2PHASE END Qualifier 19–74
PROUTIL 2PHASE MODIFY Qualifier 19–75
PROUTIL 2PHASE RECOVER Qualifier 19–76
PROUTIL BIGROW Qualifier 19–77
PROUTIL BULKLOAD Qualifier 19–78
PROUTIL BUSY Qualifier 19–79
PROUTIL CHANALYS Qualifier 19–81
PROUTIL CODEPAGE-COMPILER Qualifier 19–82
PROUTIL CONV89 Qualifier 19–83
PROUTIL CONVCHAR Qualifier 19–84
PROUTIL CONVFILE Qualifier 19–87
PROUTIL DBANALYS Qualifier 19–89
PROUTIL DBAUTHKEY Qualifier 19–90
PROUTIL DBIPCS Qualifier 19–92
PROUTIL DUMP Qualifier 19–94
PROUTIL DUMPSPECIFIED 19–97
PROUTIL ENABLELARGEFILES Qualifier 19–99
PROUTIL ENABLEPDR Qualifier 19–100
PROUTIL HOLDER Qualifier 19–101
PROUTIL IDXANALYS Qualifier 19–103
PROUTIL IDXBUILD Qualifier 19–104
PROUTIL IDXCHECK Qualifier 19–107
PROUTIL IDXCOMPACT Qualifier 19–109
PROUTIL IDXFIX Qualifier 19–111
PROUTIL IDXMOVE Qualifier 19–114
PROUTIL IOSTATS Qualifier 19–116

PROUTIL LOAD Qualifier 19–118
PROUTIL MVSCH Qualifier 19–121
PROUTIL RCODEKEY Qualifier 19–122
PROUTIL TABANALYS Qualifier 19–124
PROUTIL TABLEMOVE Qualifier 19–127
PROUTIL TRUNCATE AREA Qualifier 19–130
PROUTIL TRUNCATE BI Qualifier 19–132
PROUTIL UPDATEVST Qualifier 19–134
PROUTIL WBREAK-COMPILER Qualifier 19–135
PROUTIL WORD-RULES Qualifier 19–136
RFUTIL Utility 19–137
RFUTIL AIMAGE AIOFF 19–139
RFUTIL AIMAGE BEGIN Qualifier 19–140
RFUTIL AIMAGE END Qualifier 19–141
RFUTIL AIMAGE EXTENT EMPTY Qualifier 19–142
RFUTIL AIMAGE EXTENT FULL Qualifier 19–143
RFUTIL AIMAGE EXTENT LIST Qualifier 19–144
RFUTIL AIMAGE NEW Qualifier 19–145
RFUTIL AIMAGE SCAN Qualifier 19–146
RFUTIL AIMAGE TRUNCATE Qualifier 19–147
RFUTIL MARK BACKEDUP Qualifier 19–149
RFUTIL ROLL FORWARD Qualifier 19–150
RFUTIL ROLL FORWARD RETRY Qualifier 19–152
SQLDUMP Utility 19–154
SQLLOAD Utility 19–159
SQLSCHEMA Utility 19–164

20. Virtual System Tables 20–1
20.1 Update Access To Virtual System Tables 20–2
20.2 Virtual System Table Summaries 20–2
20.3 Progress Virtual System Table Schema Descriptions 20–8

A. Progress Monitor R&D Options A–1
Accessing PROMON R&D Options A–2
PROMON R&D Main Menu A–4
Menu Options A–7
Status Displays→ Database A–10
Status Displays→ Backup A–13
Status Displays→ Servers A–14
Status Displays→ Processes/Clients→ All Processes A–16
Status Displays→ Processes/Clients→ Blocked Clients A–20
Status Displays→ Processes/Clients→ Active Transactions A–21
Status Displays→ Processes/Clients→ Local Interactive Clients A–22
Status Displays→ Processes/Clients→ Local Batch Clients A–23

Status Displays→ Processes/Clients→ Remote Clients A–24
Status Displays→ Processes/Clients→ Background Processes A–25
Status Displays→ Files A–26
Status Displays→ Lock Table A–27
Status Displays→ Buffer Cache A–29
Status Displays→ Logging Summary A–30
Status Displays→ BI Log A–31
Status Displays→ AI Log A–32
Status Displays→ Two-phase Commit A–33
Status Displays→ Startup Parameters A–34
Status Displays→ Shared Resources A–36
Status Displays→ Shared Memory Segments A–38
Activity Displays→ Summary A–39
Activity Displays→ Servers A–43
Activity Displays→ Buffer Cache A–44
Activity Displays→ Page Writers A–46
Activity Displays→ BI Log A–48
Activity Displays→ AI Log A–50
Activity Displays→ Lock Table A–52
Activity Displays→ I/O Operations By Type A–54
Activity Displays→ I/O Operations By File A–56
Activity Displays→ Space Allocation A–57
Activity Displays→ Index A–59
Activity Displays→ Record A–60
Activity Displays→ Other A–62
Other Displays→ Performance Indicators A–63
Other Displays→ I/O Operations By Process A–65
Other Displays→ Lock Requests By User A–66
Other Displays→ Checkpoints A–67
Administrative Functions→ Resolve Limbo Transactions A–68
Administrative Functions→ Adjust Latch Options A–69
Administrative Functions→ Adjust Page Writer Options A–70
Administrative Functions→ Adjust Monitor Options A–71

B. Setting Up Progress To Utilize Compaq's NT Clusters B–1
B.1 Defining NT Clusters B–2
B.1.1 Cluster Administrator B–2
B.1.2 Failover Manager B–2
B.2 Configuring the NT Cluster System B–4
B.3 Progress Database Server Considerations and Configuration B–5
B.3.1 ProControl Command-line Interface B–6
B.4 4GL Client Considerations B–11

Index Index–1

Figures

Figure 1–1: Version 9 Structure 1–2
Figure 1–2: Physical Storage Model 1–5
Figure 1–3: Logical Storage Model 1–6
Figure 1–4: Federated Database Configuration 1–11
Figure 1–5: Distributed Database Configuration 1–12
Figure 1–6: Sample Multi-tier Configuration 1–14
Figure 1–7: Relative-path Database Creation 1–16
Figure 2–1: Matching Database and File Block Sizes 2–2
Figure 4–1: How The Schema Mover Works 4–21
Figure 8–1: Progress Recovery Mechanisms 8–2
Figure 8–2: Storing Database Files 8–6
Figure 8–3: Recovery Scenario - Day 1 8–11
Figure 8–4: Recovery Scenario - Day 2 8–12
Figure 8–5: Recovery Scenario - Day 3 8–13
Figure 8–6: Recovery Scenario - Day 4 8–14
Figure 8–7: Recovery Scenario - Day 5 8–15
Figure 11–1: After-image Extents Switching 11–2
Figure 12–1: Data Inconsistency 12–2
Figure 12–2: Two-phase Commit Algorithm 12–4
Figure 12–3: Limbo Transaction 12–5
Figure 12–4: How PROUTIL Processes React To Limbo Transactions 12–10
Figure 13–1: Sample Data Definitions File 13–5
Figure 13–2: Sample Contents File 13–15
Figure 13–3: Dictionary-created Description File Example 13–29
Figure 13–4: Copying the Starting Version Of a Database 13–31
Figure 14–1: Database I/O 14–7
Figure 14–2: Reads and Writes To Database Buffers 14–9
Figure 14–3: Evicting Buffers 14–10
Figure 14–4: Monitoring Buffer Activity 14–11
Figure 14–5: Block Table (BLKTBL) Chain 14–15
Figure 14–6: APWs and the Least Recently Used Chain 14–16
Figure 14–7: PROMON Page Writers Activity Display 14–17
Figure 14–8: PROMON BI Log Activity Display 14–19
Figure 14–9: BI Clusters At Startup 14–22
Figure 14–10: BI Clusters Over Time 14–22
Figure 14–11: PROMON AI Log Activity Display 14–28
Figure 14–12: How a Semaphore Coordinates Concurrent Access 14–34
Figure 15–1: Data Distribution Model 15–3
Figure 15–2: Data Consolidation Models 15–5
Figure 15–3: Peer-to-peer Model 15–6
Figure 15–4: Example Of Site Replication 15–9
Figure 16–1: NT Event Log Components 16–4
Figure 16–2: NT Event Detail Dialog Box 16–6

Figure 17–1: Syntax Conventions 17–2
Figure 19–1: Utility Syntax Conventions 19–2
Figure 19–2: ProControl Main Window 19–15
Figure 19–3: PROMON Utility Main Menu 19–25
Figure 19–4: Sample Output For PROMON User Control Option 19–27
Figure 19–5: Sample Output For PROMON Locking and Waiting Statistics Option . 19–30
Figure 19–6: Sample Output For PROMON Block Access Option 19–32
Figure 19–7: Sample Output For PROMON Record Locking Table Option 19–35
Figure 19–8: Sample Output For PROMON Activity Option 19–37
Figure 19–9: Sample Output For PROMON Shared Resources Option 19–41
Figure 19–10: Sample Output For PROMON Database Status Option 19–43
Figure 19–11: Sample Output For PROMON Shut Down Database Option 19–45
Figure 19–12: Sample Output For PROMON Transaction Control Option 19–46
Figure 19–13: Sample Output For PROMON Resolve Limbo Transactions Option . . 19–48
Figure 19–14: Sample Output For PROMON Coordinator Information Option 19–49
Figure 19–15: PROMON Modify Defaults Menu 19–50
Figure A–1: PROMON Database Status Display A–10
Figure A–2: PROMON Backup Status Display A–13
Figure A–3: PROMON Servers Status Display A–14
Figure A–4: PROMON Processes/Clients Status Display A–16
Figure A–5: PROMON Blocked Clients Status Display A–20
Figure A–6: PROMON Active Transactions Status Display A–21
Figure A–7: PROMON Local Interactive Clients Status Display A–22
Figure A–8: PROMON Local Batch Clients Status Display A–23
Figure A–9: PROMON Remote Clients Status Display A–24
Figure A–10: PROMON Background Processes Status Display A–25
Figure A–11: PROMON Files Status Display A–26
Figure A–12: PROMON Lock Table Status Display A–27
Figure A–13: PROMON Buffer Cache Status Display A–29
Figure A–14: PROMON Logging Summary Status Display A–30
Figure A–15: PROMON BI Log Status Display A–31
Figure A–16: PROMON AI Log Status Display A–32
Figure A–17: PROMON Two-phase Commit Status Display A–33
Figure A–18: PROMON Startup Parameters Status Display A–34
Figure A–19: PROMON Shared Resources Status Display A–36
Figure A–20: PROMON Shared Memory Segments Status Display A–38
Figure A–21: PROMON Summary Activity Display A–39
Figure A–22: PROMON Servers Activity Display A–43
Figure A–23: PROMON Buffer Cache Activity Display A–44
Figure A–24: PROMON Page Writers Activity Display A–46
Figure A–25: PROMON BI Log Activity Display A–48
Figure A–26: PROMON AI Log Activity Display A–50
Figure A–27: PROMON Lock Table Activity Display A–52
Figure A–28: PROMON I/O Operations By Type Activity Display A–54
Figure A–29: PROMON I/O Operations By File Activity Display A–56

Figure A–30: PROMON Space Allocation Activity Display A–57
Figure A–31: PROMON Index Activity Display A–59
Figure A–32: PROMON Record Activity Display A–60
Figure A–33: PROMON Other Activity Display A–62
Figure A–34: PROMON Performance Indicators Display A–63
Figure A–35: PROMON I/O Operations By Process Display A–65
Figure A–36: PROMON Lock Requests By User Display A–66
Figure A–37: PROMON Checkpoints Display A–67
Figure A–38: PROMON Resolve Limbo Transactions Menu A–68
Figure A–39: PROMON Adjust Latch Options Menu A–69
Figure A–40: PROMON Adjust Page Writer Options Menu A–70
Figure A–41: PROMON Adjust Monitor Options Menu A–71

Tables

Table 1–1: Other Database-related Files 1–4
Table 1–2: Interactive Mode Versus Batch Mode 1–10
Table 1–3: Database Configurations 1–11
Table 2–1: Formulas For Calculating Database Size 2–3
Table 2–2: Formulas For Calculating Field Storage 2–4
Table 2–3: Calculating Database Size 2–5
Table 3–1: Storage Area Types 3–2
Table 3–2: Maximum Application Data Storage Area Size 3–3
Table 3–3: Maximum Number Of Sequences 3–5
Table 3–4: Maximum Primary Recovery (BI) Area Size 3–6
Table 3–5: Maximum Number Of Connections Per Database 3–7
Table 3–6: Maximum Number Of Simultaneous Transactions 3–7
Table 3–7: Maximum Buffer Pool Size 3–7
Table 3–8: Database Name Limits 3–8
Table 3–9: SQL-92 Data Type Limits 3–10
Table 4–1: ST File Tokens and Storage Areas 4–7
Table 4–2: Reserved Storage Area Names 4–9
Table 5–1: PROSHUT Menu Options 5–14
Table 5–2: PROSHUT Parameters 5–14
Table 6–1: Backup Options 6–3
Table 6–2: Operating System Backup Utilities 6–4
Table 6–3: Backup Media Questions 6–6
Table 6–4: Availability and Transaction Loss 6–10
Table 6–5: Availability and Backup Time 6–10
Table 6–6: Availability and Recovery Time 6–11
Table 6–7: Reliability and Two-phase Commit 6–12
Table 7–1: PROREST Utility Verification Parameters 7–7
Table 8–1: Sample Low Availability Requirements 8–7
Table 8–2: Sample Moderate Availability Requirements 8–8
Table 8–3: Sample Moderate-to-High Availability Requirements 8–9
Table 8–4: Sample High Availability Requirements 8–10
Table 8–5: Utilities Used With Roll-forward Recovery 8–16
Table 8–6: Shared-memory Segment Status Fields 8–25
Table 11–1: File Tokens 11–4
Table 11–2: Operating System Backup Utilities 11–10
Table 13–1: Definitions Dumped To the Definition File 13–4
Table 13–2: Data Definitions File Trailer Values 13–6
Table 13–3: Contents File Trailer Variables 13–16
Table 13–4: Modifying the Description File 13–26
Table 14–1: Startup Parameters That Affect Memory Allocation 14–32
Table 14–2: UNIX Kernel Parameters That Affect Semaphores 14–35
Table 14–3: Factor Values 14–41
Table 16–1: The Progress Event Logging Components 16–3

Table 16–2: Event Level Values 16–3
Table 16–3: How Progress Uses Event Log Components 16–4
Table 17–1: Progress Command Components 17–2
Table 17–2: Database Startup and Shutdown Commands 17–3
Table 18–1: Server Performance Parameters 18–2
Table 18–2: Server-type Parameters 18–4
Table 18–3: Server Internationalization Parameters 18–4
Table 18–4: Server Statistics Collection Parameters 18–5
Table 18–5: Server Network Parameters 18–6
Table 18–6: Startup Parameter Categories 18–8
Table 18–7: Parameter Interpretation With Service Name (-S) 18–39
Table 19–1: Progress Command Components 19–2
Table 19–2: Type Column Values 19–27
Table 19–3: Wait Column Values 19–28
Table 19–4: Flag Values 19–36
Table 19–5: Event Types 19–38
Table 19–6: Process Types 19–40
Table 19–7: PROUTIL Utility Qualifiers 19–67
Table 19–8: RFUTIL Utility Qualifiers 19–137
Table 20–1: Progress Virtual System Tables 20–2
Table 20–2: After-image Log Activity File (_ActAILog) 20–8
Table 20–3: Before-image Log Activity File (_ActBILog) 20–9
Table 20–4: Buffer Activity File (_ActBuffer) 20–10
Table 20–5: Index Activity File (_ActIndex) 20–11
Table 20–6: Input/Output Activity File (_ActIOFile) 20–11
Table 20–7: Input/Output Type Activity File (_ActIOType) 20–12
Table 20–8: Lock Table Activity File (_ActLock) 20–13
Table 20–9: Other Activity File (_ActOther) 20–14
Table 20–10: Page Writer Activity File (_ActPWs) 20–15
Table 20–11: Record Activity File (_ActRecord) 20–16
Table 20–12: Server Activity File (_ActServer) 20–16
Table 20–13: Space Allocation Activity File (_ActSpace) 20–17
Table 20–14: Summary Activity File (_ActSummary) 20–18
Table 20–15: Area Status File (_AreaStatus) 20–19
Table 20–16: Block File (_Block) 20–20
Table 20–17: Buffer Status File (_BuffStatus) 20–20
Table 20–18: Checkpoint File (_Checkpoint) 20–21
Table 20–19: Database Connection File (_Connect) 20–22
Table 20–20: Database Status File (_DbStatus) 20–23
Table 20–21: Database File Status File (_Filelist) 20–25
Table 20–22: Index Statistics File (_IndexStat) 20–25
Table 20–23: Latch Statistics File (_Latch) 20–26
Table 20–24: License Management (_License) 20–26
Table 20–25: Lock Table Status File (_Lock) 20–27
Table 20–26: Lock Request File (_LockReq) 20–27

Table 20–27: Logging File (_Logging) 20–28
Table 20–28: Master Block File (_MstrBlk) 20–30
Table 20–29: User Connection (_MyConnection) 20–32
Table 20–30: Resource Queue Statistics File (_Resrc) 20–32
Table 20–31: Segments File (_Segments) 20–32
Table 20–32: Servers File (_Servers) 20–33
Table 20–33: Startup File (_Startup) 20–33
Table 20–34: Index and Table Statistics Range (_StatBase) 20–34
Table 20–35: Table Statistics File (_TableStat) 20–35
Table 20–36: Transaction File (_Trans) 20–35
Table 20–37: Transaction End Lock Statistics (_TxeLock) 20–36
Table 20–38: Database Input/Output File (_UserIO) 20–36
Table 20–39: Record Locking Table File (_UserLock) 20–37
Table 20–40: User Status (_UserStatus) 20–37
Table A–1: PROMON R&D Main Menu A–4
Table A–2: Database States A–10
Table A–3: Database Damaged Flags A–11
Table A–4: Integrity Flags A–11
Table A–5: Server Types A–15
Table A–6: Process Types A–16
Table A–7: Process Wait Types A–17
Table A–8: Transaction States A–21
Table A–9: Lock Table Flags A–27
Table A–10: Process Types A–42

Procedures

p-dump.p 8–27
disttran.p 12–2
StartProService.cmd B–9
StopProService.cmd B–10
StartCluDemo1.cmd B–11
repeat.p B–12
updcust.p B–13

Preface

Purpose

This manual describes Progress database administration concepts, procedures, and utilities. The procedures allow you to create and maintain your Progress databases and manage their performance.

Audience

This manual is designed as a guide and reference for Database Administrators.

Organization Of This Manual

Part I Planning

Chapter 1, “The Progress Database”

Provides an overview of the Progress database architecture.

Chapter 2, “Administrative Planning”

Presents guidelines that help you make initial decisions about how to administer and maintain a Progress database.

Chapter 3, “Progress Database Limits”

Lists the Progress database limits you need to know when configuring a Progress database.

Part II Administration

Chapter 4, “Creating and Deleting Databases”

Describes how to create and delete Progress databases.

Chapter 5, “Starting Up and Shutting Down”

Describes the commands required to start up and shut down a Progress database.

Chapter 6, “Backup and Recovery Strategies”

Describes backup and recovery strategies.

Chapter 7, “Backing Up a Database”

Describes how to use the Progress PROBKUP utility to back up a database.

Chapter 8, “Recovering a Database”

Describes how to use the Progress PROREST utility to restore a database.

Chapter 9, “Maintaining Database Structure”

Describes methods to manage the database structure and alter it as necessary to improve storage and performance.

Chapter 10, “Maintaining Security”

Describes how Progress implements database security, including assigning user IDs, designating database administrators.

Chapter 11, “After-Imaging”

Describes after-imaging and how to use it for data recovery. Also, describes how to implement after-imaging with after image extents.

Chapter 12, “Using Two-phase Commit”

Describes two-phase commit and how to use it to ensure database integrity.

Chapter 13, “Dumping and Loading”

Describes how to dump and load databases, including tables, indexes, and sequences.

Chapter 14, “Managing Performance”

Describes how to tune database performance.

Chapter 15, “Replicating Data”

Describes replication schemes and how to implement log-based replication.

Chapter 16, “Using the Event Log”

Describes the logging of significant events.

Part III Reference

Chapter 17, “Startup and Shutdown Commands”

Describes Progress commands for starting up and shutting down sessions and processes.

Chapter 18, “Database Startup Parameters”

Describes the Progress database startup parameters.

Chapter 19, “Database Administration Utilities”

Describes the Progress database administration utilities.

Chapter 20, “Virtual System Tables”

Describes the Virtual System Tables that allow 4GL and SQL-92 applications to examine the status of a database and monitor its performance.

Appendix A, “Progress Monitor R&D Options”

Describes the Progress Monitor R&D Options.

Appendix B, “Setting Up Progress To Utilize Compaq’s NT Clusters”

Describes how to set up and use NT clusters.

How To Use This Manual

This book is organized into three distinct but complementary parts:

- Part 1, “Planning,” provides an overview and introduction to the Progress database. Read this part first, beginning with Chapter 1, “The Progress Database.”
- Part 2, “Administration,” describes the procedures a Database Administrator uses to manage a database in a flexible business environment. Each chapter discusses a particular administrative activity, such as Chapter 6, “Backing Up and Restoring.”
- Part 3, “Reference,” describes the Progress database commands, startup parameters, utilities, and system tables in alphabetical order. Refer to the chapters in Part 3 when you need to access specific descriptive information, such as the syntax of an administration utility.

Typographical Conventions

This manual uses the following typographical conventions:

- **Bold typeface** indicates:
  - Commands or characters that the user types
  - That a word carries particular weight or emphasis
- *Italic typeface* indicates:
  - Progress variable information that the user supplies
  - New terms
  - Titles of complete publications
- `Monospaced typeface` indicates:
  - Code examples
  - System output
  - Operating system filenames and pathnames

The following typographical conventions are used to represent keystrokes:

- Small capitals are used for Progress key functions and generic keyboard keys.

  END-ERROR, GET, GO
  ALT, CTRL, SPACEBAR, TAB

- When you have to press a combination of keys, they are joined by a dash. You press and hold down the first key, then press the second key.

  CTRL-X

- When you have to press and release one key, then press another key, the key names are separated with a space.

  ESCAPE H
  ESCAPE CURSOR-LEFT

Syntax Notation

The syntax for each component follows a set of conventions:

- Uppercase words are keywords. Although they are always shown in uppercase, you can use either uppercase or lowercase when using them in a procedure.

  In this example, ACCUM is a keyword:

  SYNTAX

  ```
  ACCUM aggregate expression
  ```

- Italics identify options or arguments that you must supply. These options can be defined as part of the syntax or in a separate syntax identified by the name in italics. In the ACCUM function above, the *aggregate* and *expression* options are defined with the syntax for the ACCUM function in the *Progress Language Reference*.

- You must end all statements (except for DO, FOR, FUNCTION, PROCEDURE, and REPEAT) with a period. DO, FOR, FUNCTION, PROCEDURE, and REPEAT statements can end with either a period or a colon, as in this example:

  ```
  FOR EACH Customer:
     DISPLAY Name.
  END.
  ```

- Square brackets (**[ ]**) around an item indicate that the item, or a choice of one of the enclosed items, is optional.

  In this example, STREAM *stream*, UNLESS-HIDDEN, and NO-ERROR are optional:

  SYNTAX

  ```
  DISPLAY [ STREAM stream ] [ UNLESS-HIDDEN ] [ NO-ERROR ]
  ```

  In some instances, square brackets are not a syntax notation, but part of the language.

  For example, this syntax for the INITIAL option uses brackets to bound an initial value list for an array variable definition. In these cases, normal text brackets ([ ]) are used:

  SYNTAX

  ```
  INITIAL [ constant [ , constant ] ... ]
  ```

  NOTE: The ellipsis (**...**) indicates repetition, as shown in a following description.

- Braces (**{ }**) around an item indicate that the item, or a choice of one of the enclosed items, is required.

  In this example, you must specify the items BY and *expression* and can optionally specify the item DESCENDING, in that order:

  SYNTAX

  ```
  { BY expression [ DESCENDING ] }
  ```

  In some cases, braces are not a syntax notation, but part of the language.

  For example, a called external procedure must use braces when referencing arguments passed by a calling procedure. In these cases, normal text braces ({ }) are used:

  SYNTAX

  ```
  { &argument-name }
  ```

- A vertical bar (**|**) indicates a choice.

  In this example, EACH, FIRST, and LAST are optional, but you can only choose one:

  SYNTAX

  ```
  PRESELECT [ EACH | FIRST | LAST ] record-phrase
  ```

  In this example, you must select one of *logical-name* or *alias*:

  SYNTAX

  ```
  CONNECTED ( { logical-name | alias } )
  ```

- Ellipses (**. . .**) indicate that you can choose one or more of the preceding items. If a group of items is enclosed in braces and followed by ellipses, you must choose one or more of those items. If a group of items is enclosed in brackets and followed by ellipses, you can optionally choose one or more of those items.

  In this example, you must include two expressions, but you can optionally include more. Note that each subsequent expression must be preceded by a comma:

  SYNTAX

  ```
  MAXIMUM ( expression , expression [ , expression ] ... )
  ```

  In this example, you must specify MESSAGE, then at least one of *expression* or SKIP, but any additional number of *expression* or SKIP is allowed:

  SYNTAX

  ```
  MESSAGE { expression | SKIP [ (n) ] } ...
  ```

In this example, you must specify {*include-file*, then optionally any number of *argument* or &*argument-name* = "*argument-value*", and then terminate with }:

SYNTAX

```
{ include-file
    [ argument | &argument-name = "argument-value" ] ... }
```

- In some examples, the syntax is too long to place in one horizontal row. In such cases, **optional** items appear individually bracketed in multiple rows in order, left-to-right and top-to-bottom. This order generally applies, unless otherwise specified. **Required** items also appear on multiple rows in the required order, left-to-right and top-to-bottom. In cases where grouping and order might otherwise be ambiguous, braced (required) or bracketed (optional) groups clarify the groupings.

  In this example, WITH is followed by several optional items:

  SYNTAX

```
WITH [ ACCUM max-length ] [ expression DOWN ]
  [ CENTERED ] [ n COLUMNS ] [ SIDE-LABELS ]
  [ STREAM-IO ]
```

  In this example, ASSIGN requires one of two choices: either one or more of *field*, or one of *record*. Other options available with either *field* or *record* are grouped with braces and brackets. The open and close braces indicate the required order of options:

  SYNTAX

```
ASSIGN {   { [ FRAME frame ]
               { field [ = expression ] }
               [ WHEN expression ]
           } ...
         | { record [ EXCEPT field ... ] }
       }
```

Example Procedures

This manual provides example procedures that illustrate syntax and concepts. Examples use the following conventions:

- They appear in boxes with borders.
- If they are available online, the name of the procedure appears above the left corner of the box and starts with a prefix associated with the manual that references it, as follows:
  - `e-` — *Progress External Program Interfaces*, for example, `e-ddeex1.p`
  - `lt-` — *Progress Language Tutorials*, for example, `lt-05-s3.p`
  - `p-` — *Progress Programming Handbook*, for example, `p-br01.p`
  - `r-` — *Progress Language Reference*, for example, `r-dynbut.p`

  If the name does not start with a listed prefix, the procedure is not available online.
- If they are not available online, they compile as shown, but might not execute for lack of completeness.

Accessing Files In Procedure Libraries

Documentation examples are stored in procedure libraries, `prodoc.pl` and `prohelp.pl`, in the `src` directory where Progress is installed.

You **must** first create all subdirectories required by a library before attempting to extract files from the library. You can see what directories and subdirectories a library needs by using the PROLIB -list command to view the contents of the library. See the *Progress Client Deployment Guide* for more details on the PROLIB utility.

Creating a Listing Of the Procedure Libraries

Creating a listing of the source files from a procedure library involves running PROENV to set up your Progress environment, and running PROLIB.

1 ♦ From the Control Panel or the Progress Program Group, double-click the Proenv icon.

2 ♦ The Proenv Window appears, with the proenv prompt.

Running Proenv sets the DLC environment variable to the directory where you installed Progress (by default, `C:\Program Files\Progress`). Proenv also adds the DLC environment variable to your PATH environment variable and adds the bin directory (`PATH=%DLC%;%DLC%\bin;%PATH%`).

3 ♦ Enter the following command at the proenv prompt to create the text file `prodoc.txt` which contains the file listing for the prodoc.pl library.

```
PROLIB %DLC%\src\prodoc.pl -list > prodoc.txt
```

Extracting Source FilesFrom Procedure Libraries On Windows Platforms

Extracting source files from a procedure library involves running PROENV to set up your Progress environment, creating the directory structure for the files you want to extract, and running PROLIB.

1 ♦ From the Control Panel or the Progress Program Group, double-click the Proenv icon.

2 ♦ The Proenv Window appears, with the proenv prompt.

Running Proenv sets the DLC environment variable to the directory where you installed Progress (by default, `C:\Program Files\Progress`). Proenv also adds the DLC environment variable to your PATH environment variable and adds the bin directory (`PATH=%DLC%;%DLC%\bin;%PATH%`).

3 ♦ Enter the following command at the proenv prompt to create the `prodoc` directory in your Progress working directory (by default, `C:\Progress\Wrk`):

```
MKDIR prodoc
```

4 ♦ Create the `langref` directory under `prodoc`:

```
MKDIR prodoc\langref
```

5 ♦ To extract all examples in a procedure library directory, run the PROLIB utility. Note that you must use double quotes because “Program Files” contains an embedded space:

```
PROLIB "%DLC%\src\prodoc.pl" -extract prodoc\langref\*.*
```

PROLIB extracts all examples into `prodoc\langref`.

To extract one example, run PROLIB and specify the file that you want to extract as it is stored in the procedure library:

```
PROLIB "%DLC%\src\prodoc.pl" -extract prodoc/langref/r-syshlp.p
```

PROLIB extracts `r-syshlp.p` into `prodoc\langref`.

Extracting Source Files From Procedure Libraries On UNIX Platforms

To extract `p-wrk1.p` from `prodoc.pl`, a procedure library, follow these steps at the UNIX system prompt:

1 ♦ Run the PROENV utility:

```
install-dir/dlc/bin/proenv
```

Running proenv sets the DLC environment variable to the directory where you installed Progress (by default, `/usr/dlc`). The proenv utility also adds the bin directory under the DLC environment variable to your PATH environment variable (`PATH=$DLC/bin:$PATH`).

2 ♦ At the proenv prompt, create the `prodoc` directory in your Progress working directory:

```
mkdir prodoc
```

3 ♦ Create the `proghand` directory under `prodoc`:

```
mkdir prodoc/proghand
```

4 ♦ To extract all examples in a procedure library directory, run the PROLIB utility:

```
prolib $DLC/src/prodoc.pl -extract prodoc/proghand/*.*
```

PROLIB extracts all examples into `prodoc/proghand`.

To extract one example, run PROLIB and specify the file that you want to extract as it is stored in the procedure library:

```
prolib $DLC/src/prodoc.pl -extract prodoc/proghand/p-wrk-1.p
```

PROLIB extracts `p-wrk-1.p` into `prodoc/proghand`.

Progress Messages

Progress displays several types of messages to inform you of routine and unusual occurrences:

- Execution messages inform you of errors encountered while Progress is running a procedure (for example, if Progress cannot find a record with a specified index field value).
- Compile messages inform you of errors found while Progress is reading and analyzing a procedure prior to running it (for example, if a procedure references a table name that is not defined in the database).
- Startup messages inform you of unusual conditions detected while Progress is getting ready to execute (for example, if you entered an invalid startup parameter).

After displaying a message, Progress proceeds in one of several ways:

- Continues execution, subject to the error-processing actions that you specify, or that are assumed, as part of the procedure. This is the most common action taken following execution messages.
- Returns to the Progress Procedure Editor so that you can correct an error in a procedure. This is the usual action taken following compiler messages.
- Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.
- Terminates the current session.

Progress messages end with a message number in parentheses. In this example, the message number is 200:

```
** Unknown table name table. (200)
```

On the Windows platform, use Progress online help to get more information about Progress messages. Many Progress tools include the following Help menu options to provide information about messages:

- Choose Help→Recent Messages to display detailed descriptions of the most recent Progress message and all other messages returned in the current session.
- Choose Help→Messages, then enter the message number to display a description of any Progress message. (If you encounter an error that terminates Progress, make a note of the message number before restarting.)
- In the Procedure Editor, press the **HELP** key (**F2** or **CTRL-W**).

On the UNIX platform, use the Progress PRO command to start a single-user mode character Progress client session and view a brief description of a message by providing its number. Follow these steps:

1 ♦ Start the Progress Procedure Editor:

```
install-dir/dlc/bin/pro
```

2 ♦ Press **F3** to access the menu bar, then choose Help→Messages.

3 ♦ Type the message number, and press **ENTER**. Details about that message number appear.

4 ♦ Press **F4** to close the message, press **F3** to access the Procedure Editor menu, and choose File→Exit.

Other Useful Documentation

This section lists Progress Software Corporation documentation that you might find useful. Unless otherwise specified, these manuals support both Windows and Character platforms and are provided in electronic documentation format on CD-ROM.

Getting Started

Progress Electronic Documentation Installation and Configuration Guide (Hard copy only)

A booklet that describes how to install the Progress EDOC viewer and collection on UNIX and Windows.

Progress Installation and Configuration Guide Version 9 for UNIX

A manual that describes how to install and set up Progress Version 9.1 for the UNIX operating system.

Progress Installation and Configuration Guide Version 9 for Windows

A manual that describes how to install and set up Progress Version 9.1 for all supported Windows and Citrix MetaFrame operating systems.

Progress Version 9 Product Update Bulletin

A bulletin that provides a list of new and changed features by release, a list and description of changes to documentation by release, and critical information about product changes that might require changes to existing code and configurations.

This bulletin also provides information about where to go for detailed information about the new and changed features and documentation.

Progress Application Development Environment — Getting Started (Windows only)

A practical guide to graphical application development within the Progress Application Development Environment (ADE). This guide includes an overview of the ADE and its tools, an overview of Progress SmartObject technology, and tutorials and exercises that help you better understand SmartObject technology and how to use the ADE to develop applications.

Progress Language Tutorial for Windows and *Progress Language Tutorial for Character*

Platform-specific tutorials designed for new Progress users. The tutorials use a step-by-step approach to explore the Progress application development environment using the 4GL.

Progress Master Glossary for Windows and *Progress Master Glossary for Character* (EDOC only)

Platform-specific master glossaries for the Progress documentation set. These books are in electronic format only.

Progress Master Index and Glossary for Windows and *Progress Master Index and Glossary for Character* (Hard copy only)

Platform-specific master indexes and glossaries for the Progress hard-copy documentation set.

Progress Startup Command and Parameter Reference

A reference manual that describes the Progress startup and shutdown commands that you use at the command line, and the startup parameters that you use for Progress processes. This guide also provides information about parameter usage and parameter files.

Welcome to Progress (Hard copy only)

A booklet that explains how Progress software and media are packaged. An icon-based map groups the documentation by functionality, providing an overall view of the documentation set. *Welcome to Progress* also provides descriptions of the various services Progress Software Corporation offers.

Development Tools

Progress ADM 2 Guide

A guide to using the Application Development Model, Version 2 (ADM 2) application architecture to develop Progress applications. It includes instructions for building and using Progress SmartObjects.

Progress ADM 2 Reference

A reference for the Application Development Model, Version 2 (ADM 2) application. It includes descriptions of ADM 2 functions and procedures.

Progress AppBuilder Developer's Guide (Windows only)

A programmer's guide to using the Progress AppBuilder visual layout editor. AppBuilder is a Rapid Application Development (RAD) tool that can significantly reduce the time and effort required to create Progress applications.

Progress Basic Database Tools (Character only; information for Windows is in online help)

A guide for the Progress Database Administration tools, such as the Data Dictionary.

Progress Basic Development Tools (Character only; information for Windows is in online help)

A guide for the Progress development toolset, including the Progress Procedure Editor and the Application Compiler.

Progress Debugger Guide

A guide for the Progress Application Debugger. The Debugger helps you trace and correct programming errors by allowing you to monitor and modify procedure execution as it happens.

Progress Help Development Guide (Windows only)

A guide that describes how to develop and integrate an online help system for a Progress application.

Progress Translation Manager Guide (Windows only)

A guide that describes how to use the Progress Translation Manager tool to manage the entire process of translating the text phrases in Progress applications.

Progress Visual Translator Guide (Windows only)

A guide that describes how to use the Progress Visual Translator tool to translate text phrases from procedures into one or more spoken languages.

Reporting Tools

Progress Report Builder Deployment Guide (Windows only)

An administration and development guide for generating Report Builder reports using the Progress Report Engine.

Progress Report Builder Tutorial (Windows only)

A tutorial that provides step-by-step instructions for creating eight sample Report Builder reports.

Progress Report Builder User's Guide (Windows only)

A guide for generating reports with the Progress Report Builder.

Progress Results Administration and Development Guide (Windows only)

A guide for system administrators that describes how to set up and maintain the Results product in a graphical environment. This guide also describes how to program, customize, and package Results with your own products. In addition, it describes how to convert character-based Results applications to graphical Results applications.

Progress Results User's Guide for Windows and *Progress Results User's Guide for Unix*

Platform-specific guides for users with little or no programming experience that explain how to query, report, and update information with Results. Each guide also helps advanced users and application developers customize and integrate Results into their own applications.

4GL

Building Distributed Applications Using the Progress AppServer

A guide that provides comprehensive information about building and implementing distributed applications using the Progress AppServer. Topics include basic product information and terminology, design options and issues, setup and maintenance considerations, 4GL programming details, and remote debugging.

Progress External Program Interfaces

A guide to the external programming interfaces supported by Progress. This manual covers the Host Language Call (HLC) Interface, the system clipboard, named pipes, shared libraries and DLLS, Windows Dynamic Data Exchange (DDE), COM objects, ActiveX Automation, ActiveX controls, sockets, XML, SAX, and the SonicMQ 4GL Adapter.

Progress Internationalization Guide

A guide to developing Progress applications for markets worldwide. The guide covers both internationalization—writing an application so that it adapts readily to different locales (languages, cultures, or regions)—and localization—adapting an application to different locales.

Progress Language Reference

A three-volume reference set that contains extensive descriptions and examples for each statement, phrase, function, operator, widget, attribute, method, and event in the Progress language.

Progress on the Web

A manual that describes how to use the new WebClient, AppServer Internet Adapter, SmartObjects, and SonicMQ Adapter to create applications tailored for Internet, intranet, and extranet environments.

Progress Programming Handbook

A two-volume handbook that details advanced Progress programming techniques.

Database

Progress Database Design Guide

A guide that uses a sample database and the Progress Data Dictionary to illustrate the fundamental principles of relational database design. Topics include relationships, normalization, indexing, and database triggers.

DataServers

Progress DataServer Guides

These guides describe how to use the DataServers to access non-Progress databases. They provide instructions for building the DataServer modules, a discussion of programming considerations, and a tutorial.

Each DataServer has its own guide as follows:

- *Progress/400 Product Guide*
- *Progress DataServer for Microsoft SQL Server Guide*
- *Progress DataServer for ODBC Guide*
- *Progress DataServer for ORACLE Guide*

MERANT ODBC Branded Driver Reference

The Enterprise DataServer for ODBC includes MERANT ODBC drivers for all the supported data sources. For configuration information, see the MERANT documentation, which is available as a PDF file in *installation-path*\odbc. To read this file you must have the Adobe Acrobat Reader Version installed on your system. If you do not have the Adobe Acrobat Reader, you can download it from the Adobe Web site at: http://www.adobe.com/products/acrobat/readstep.html.

SQL-89/Open Access

Progress Embedded SQL-89 Guide and Reference

A guide to Progress Embedded SQL-89 for C, including step-by-step instructions on building ESQL-89 applications and reference information on all Embedded SQL-89 Preprocessor statements and supporting function calls. This guide also describes the relationship between ESQL-89 and the ANSI standards upon which it is based.

Progress Open Client Developer's Guide

A guide that describes how to write, build, and deploy Java and ActiveX applications, and Java applets that run as clients of the Progress AppServer. This guide includes information about how to expose the AppServer as a set of Java classes or as an ActiveX server, and how to choose an Open Client distribution package for run time.

Progress SQL-89 Guide and Reference

A user guide and reference for programmers who use interactive Progress/SQL-89. It includes information on all supported SQL-89 statements, SQL-89 Data Manipulation Language components, SQL-89 Data Definition Language components, and supported Progress functions.

SQL-92

Progress Embedded SQL-92 Guide and Reference

A guide to Progress Embedded SQL-92 for C, including step-by-step instructions for building ESQL-92 applications and reference information about all Embedded SQL-92 Preprocessor statements and supporting function calls. This guide also describes the relationship between ESQL-92 and the ANSI standards upon which it is based.

Progress JDBC Driver Guide

A guide to the Java Database Connectivity (JDBC) interface and the Progress SQL-92 JDBC driver. It describes how to set up and use the driver and details the driver's support for the JDBC interface.

Progress ODBC Driver Guide

A guide to the ODBC interface and the Progress SQL-92 ODBC driver. It describes how to set up and use the driver and details the driver's support for the ODBC interface.

Progress SQL-92 Guide and Reference

A user guide and reference for programmers who use Progress SQL-92. It includes information on all supported SQL-92 statements, SQL-92 Data Manipulation Language components, SQL-92 Data Definition Language components, and Progress functions. The guide describes how to use the Progress SQL-92 Java classes and how to create and use Java stored procedures and triggers.

Deployment

Progress Client Deployment Guide

A guide that describes the client deployment process and application administration concepts and procedures.

Progress Developer's Toolkit

A guide to using the Developer's Toolkit. This guide describes the advantages and disadvantages of different strategies for deploying Progress applications and explains how you can use the Toolkit to deploy applications with your selected strategy.

Progress Portability Guide

A guide that explains how to use the Progress toolset to build applications that are portable across all supported operating systems, user interfaces, and databases, following the Progress programming model.

WebSpeed

Getting Started with WebSpeed

Provides an introduction to the WebSpeed Workshop tools for creating Web applications. It introduces you to all the components of the WebSpeed Workshop and takes you through the process of creating your own Intranet application.

WebSpeed Installation and Configuration Guide

Provides instructions for installing WebSpeed on Windows and UNIX systems. It also discusses designing WebSpeed environments, configuring WebSpeed Brokers, WebSpeed Agents, and the NameServer, and connecting to a variety of data sources.

WebSpeed Developer's Guide

Provides a complete overview of WebSpeed and the guidance necessary to develop and deploy WebSpeed applications on the Web.

WebSpeed Product Update Bulletin

A booklet that provides a brief description of each new feature of the release. The booklet also explains where to find more detailed information in the documentation set about each new feature.

Welcome to WebSpeed (Hard copy only)

A booklet that explains how WebSpeed software and media are packaged. *Welcome to WebSpeed!* also provides descriptions of the various services Progress Software Corporation offers.

Reference

Pocket Progress (Hard copy only)

A reference that lets you quickly look up information about the Progress language or programming environment.

Pocket WebSpeed (Hard copy only)

A reference that lets you quickly look up information about the SpeedScript language or the WebSpeed programming environment.

PART I

Planning

The Progress Database

Administrative Planning

Progress Database Limits

1

The Progress Database

When administering a Progress database, it is important to understand its architecture and the configuration options it supports. This chapter presents an overview of the Progress Version 9 database. Specifically, it contains the following sections:

- Progress Database Architecture
- Storage Design Overview
- Determining Configuration Variables
- Operating System Resources
- Multi-threaded Architecture
- Multi-tier and Cluster Configurations
- Self-service and Network Clients
- Relative- and Absolute-path Databases

1.1 Progress Database Architecture

The three Progress Version 9 databases (Enterprise, Workgroup, and Personal) contain not only data but also information about their schema. Figure 1–1 illustrates the components of a Progress Version 9 Database.

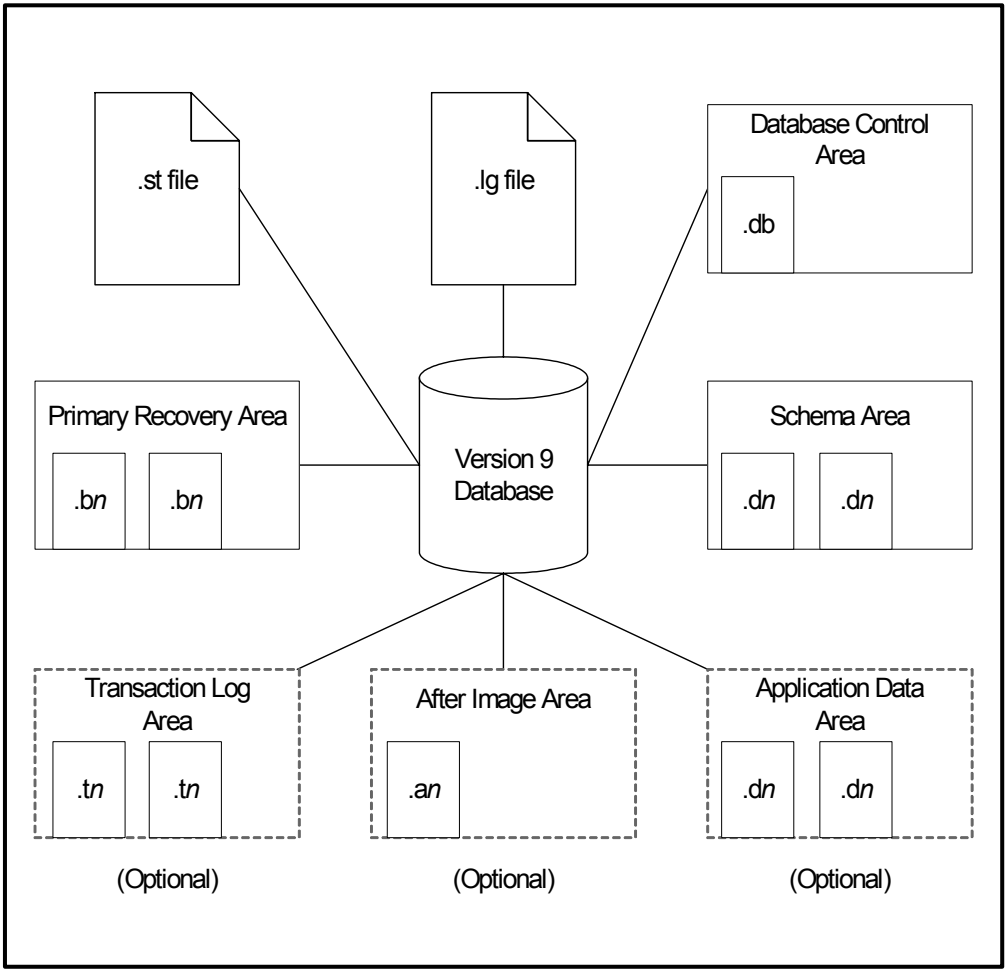


Figure 1–1: Version 9 Structure

As shown in Figure 1–1, a typical Progress Version 9 database consists of:

- A structure description (.st) file, which defines the database's structure. The .st file is a text file with an .st filename extension. The administration utility PROSTRCT CREATE uses the information in the .st file to create the database's areas and extents. See Chapter 4, "Creating and Deleting Databases," for detailed information about structure description files.

- An event log file, which is a text file with a .lg filename extension. The .lg file contains a history of significant database events, such as startup and shutdown.

- One database (.db) control area, which contains a database structure extent (a binary file with a .db filename extension). The control area and its .db file act as a table of contents, for the database engine, listing the name and location of every area and extent in the database.

- One primary recovery (before image) area, which contains one or more extents with a .b*n* filename extension. The .bi file stores notes about data changes. In the event of hardware failure, the database engine uses these notes to undo any incomplete transactions and maintain data integrity.

- One schema area, which contains at least one variable-length extent with a .d*n* filename extension. The schema area contains the master and sequence blocks, as well as schema tables and indexes. If no application data areas are created, the schema area contains all user data.

- Optionally, one transaction log (.tl) area when two-phase commit is in use. It contains one or more fixed-length extents with the .t*n* filename extension; variable-length extents are not allowed. The transaction log lists committed two-phase commit transactions.

- Optionally, one after-image (.ai) area when after-imaging is enabled. The after-image area can contain many fixed-length extents and/or one variable-length extent with the .a*n* filename extension. In the event of hardware failure, the database engine uses the .ai file and the most recent backup to reconstruct the database.

  NOTE: After-imaging is only available on the Progress Enterprise database.

- Optionally, one or more application data areas each containing extents with the .d*n* filename extension. Application data areas contain tables, indexes, and other objects.

NOTE: Throughout this guide, the word database means collectively, the database, event log, transaction log, schema files and recovery files. You should treat these files as an indivisible unit. For example, the phrase “back up the database” means “back up the database data, .lg, .d*n*, .tl, .ai, and .bi files together.” This is an important concept to remember as you read through this guide.

1.1.1 Other Database-related Files

While maintaining your database, you might encounter files with the extensions listed in Table 1–1.

Table 1–1: Other Database-related Files

File	Description
.bd	Binary dump file (table-based)
.cf	Schema cache file
.cp	Binary compiled code page information file
.dfsql	SQL-92 data definition file
.dsql	SQL-92 table dump file
.d	4GL table dump file
.df	4GL data definition file
.fd	4GL bulk loader description file
.lic	License file
.lk	Lock file
.rpt	License usage report file

1.2 Storage Design Overview

The storage design of a Progress Version 9 database is divided into a physical model and a logical model. You manipulate the physical storage model through 4GL, SQL-92, and database administration utility interfaces. Figure 1–2 shows how areas can be stored on and even span different file slices. Note that though an area's extents can contain many disk files, an extent can only be associated with one storage area.

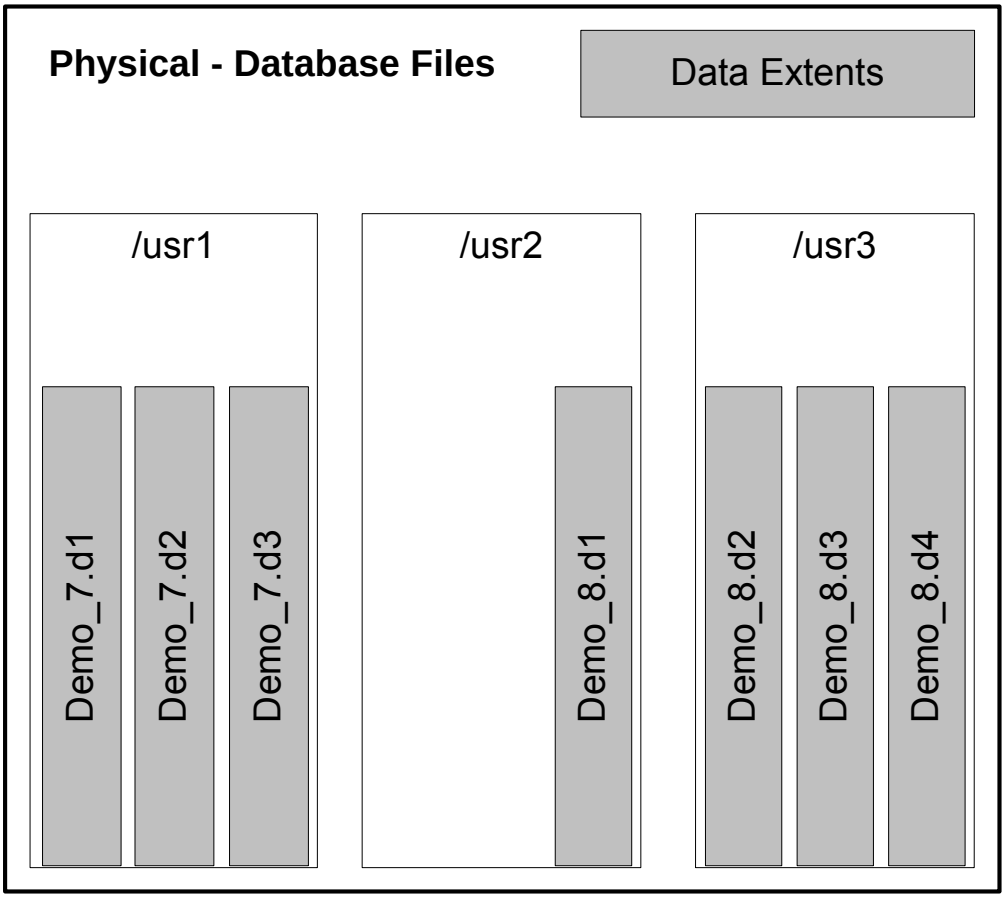


Figure 1–2: Physical Storage Model

The logical storage model can be manipulated through 4GL and SQL-92 interfaces. Logical database objects are described in the database schema and include tables, indexes, and sequences. You update these objects through 4GL and SQL-92 language statements. Figure 1–3 illustrates how logical objects can span physical extents.

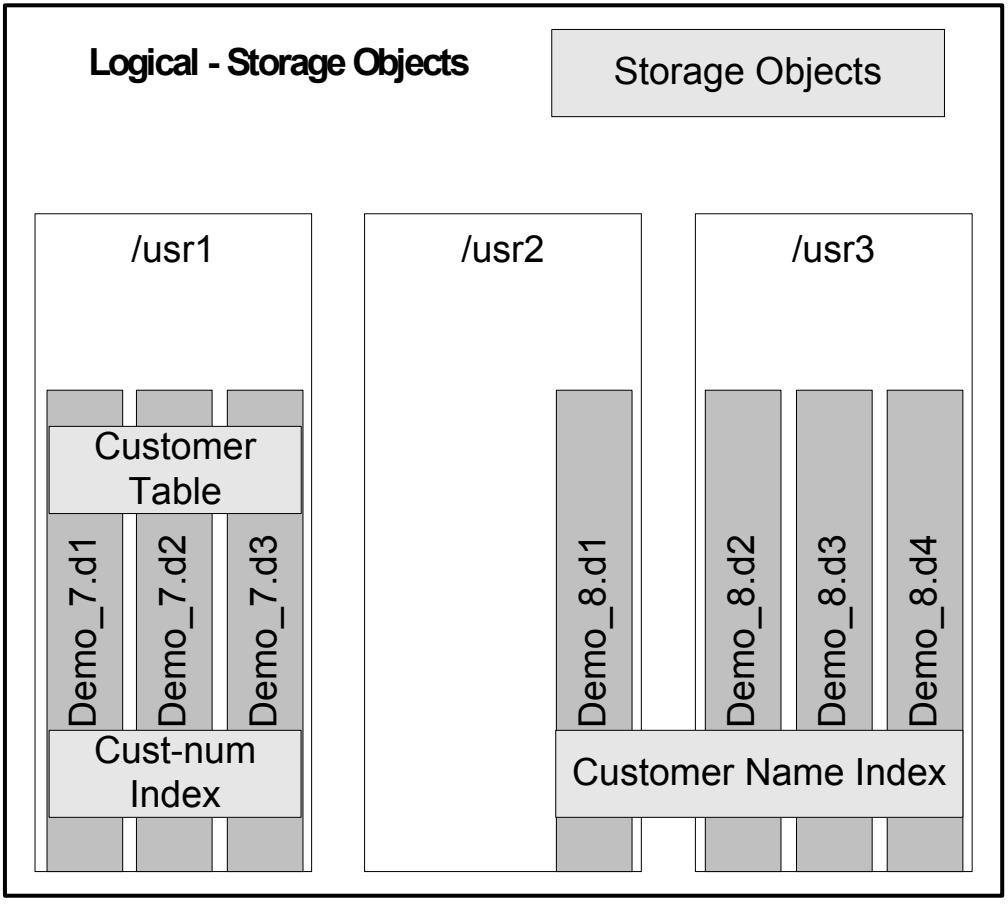


Figure 1–3: Logical Storage Model

The following sections describe the three components of the Progress Version 9 storage model.

1.2.1 Storage Objects

Storage objects are collections of disk space associated with one or more specific database objects. Each storage object is an instance of a database object described in the schema. Types of storage objects include tables and indexes.

1.2.2 Extents

Extents make it possible for a Progress Version 9 database to extend across more than one file system or physical volume. Extents are disk files or UNIX raw partitions that store physical blocks of database objects.

There are two types of extents: fixed-length and variable-length. With *fixed-length extents* you control how much disk space each extent uses by defining in the .st file the size of the extent in kilobytes. *Variable-length extents* do not have a predefined length and can continue to grow until they use all available space on a disk.

See Chapter 4, "Creating and Deleting Databases," for information on defining both fixed- and variable-length extents in the .st file.

1.2.3 Storage Areas

Storage areas consist of one or more extents and are the largest physical units of a database. With storage areas, you have physical control over the location of database objects: you can place each database object in its own storage area, place many database objects in a single storage area, and place objects of different types in the same storage area. See Chapter 2, "Administrative Planning," for information on managing storage areas. Though you can extend a table or index across multiple extents, you cannot split them across storage areas.

Certain storage areas have restrictions on the types of extents they support. The transaction log storage area, used for two-phase commit, uses only fixed-length extents but can use more than one. The other storage areas can use many extents but only one variable-length extent, which must be the last extent.

Storage areas are identified by their names. The number and type of storage areas used varies from database to database; however, all Progress databases must contain a control area, a schema area, and a primary recovery area. These three mandatory storage areas are detailed below, along with the optional areas.

Control Area

The control area contains only one variable-length extent: the database structure extent, which is a binary file with a .db extension. The .db file contains the _area table and the _area-extent table, which list the name of every area in the database, as well as the location and size of each extent.

Schema Area

The schema area can contain as many fixed-length extents as needed; however, every schema area should have a variable-length extent as its last extent. The schema area stores all database system and user information, and any objects not assigned to another area. If you choose not to create any optional application data areas, the schema area contains all of the database's objects and sequences.

Primary Recovery Area

The primary recovery area can contain as many fixed-length extents as needed, as long as the last extent is of variable length. The primary recovery area is also called the before-image area because the .b*n* files of its extents record data changes. In the event of a hardware failure, the Progress Restore (PROREST) utility uses the contents of the .b*n* files to restore the database.

Application Data Area

The application data storage area contains all application-related database objects. Defining more than one application data area allows you to improve database performance by storing different objects on different disks. Each application data area contains one or more extents with a .d*n* extension.

Transaction Log Area

The transaction log area is required if two-phase commit is used. This area contains one or more fixed-length extents with the .t*n* filename extension; variable-length extents are not allowed. See Chapter 12, "Using Two-phase Commit," for more information about the transaction log area.

1.3 Determining Configuration Variables

The Progress Version 9 database architecture and its wide range of supported system platforms allow flexible configuration options. A very simple configuration might support a single user on a PC system. A complex configuration might include heterogeneous, networked systems supporting hundreds of users accessing multiple databases, including non-Progress databases.

Your business needs determine your choices for the following variables:

- System platform
- User connection modes
- Location of databases
- Database configurations

The following sections explain how these variables affect your configuration.

1.3.1 System Platform

The Progress Version 9 database provides a multi-threaded database server that can service multiple network clients. Each server handles many simultaneous requests from clients. The server processes simple requests as a single operation to provide rapid responses; it divides complex requests into smaller tasks to minimize the impact on other users of the database.

Certain administration tasks, such as setting up user-interface environments and performance tuning, differ among operating systems.

1.3.2 Connection Modes

Progress databases run in one of two *connection modes*: single-user or multi-user. Connection modes control how many users can access a database simultaneously.

Single-user Mode

A database running in single-user mode allows one user to access a specified database at a time. If another user is already accessing a database, you cannot connect to that database from a different Progress session. The database engine uses a lock file (.lk) to lock out other users.

Running a database in single-user mode is helpful when you perform system administration tasks that require exclusive access to the database.

Multi-user Mode

A database running in multi-user mode enables more than one user to access it simultaneously. A broker coordinates all the database requests from all the users using a single database. For example, the broker process locks the database to prevent any other broker or single-user process from opening it. Also, all users must request access from the broker to connect to the database.

Both single-user and multi-user processes can run in either interactive or batch mode. Table 1–2 compares these modes.

Table 1–2: Interactive Mode Versus Batch Mode

Mode	Description
Interactive mode	The user interacts directly with the database.
Batch mode	Processing occurs without user interaction. Batch mode is convenient for large-scale database updates or procedures that can run unattended (at night, for example). However, multi-user batch jobs can degrade the response time for interactive users.

1.3.3 Database Location

A Progress Version 9 database can be either local or remote. A database located on the same machine as the application session is local. A database located on a machine that is networked to the machine containing the application session is remote.

1.3.4 Database Configurations

When the user connects to more than one database, there are two basic configurations: federated and distributed. Table 1–3 compares these configurations. Figure 1–4 and Figure 1–5 show the two configurations.

Table 1–3: Database Configurations

Database Configuration	Description
Federated	All databases are local. Figure 1–4 shows a sample federated database configuration.
Distributed	One or more of the databases reside on one or more remote machines in the network, and Progress sessions connect to the database using a single networking protocol. Figure 1–5 shows a sample distributed configuration.

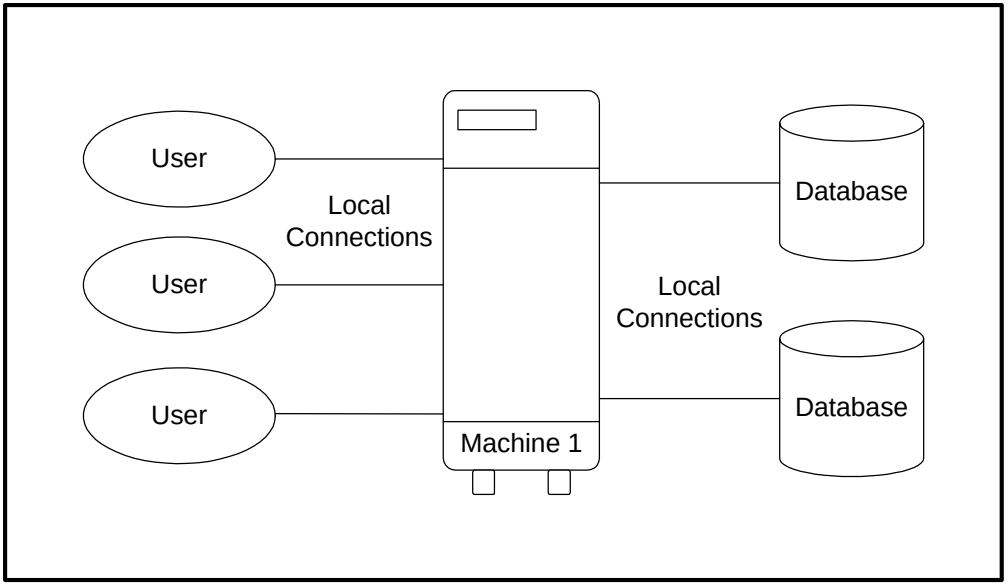


Figure 1–4: Federated Database Configuration

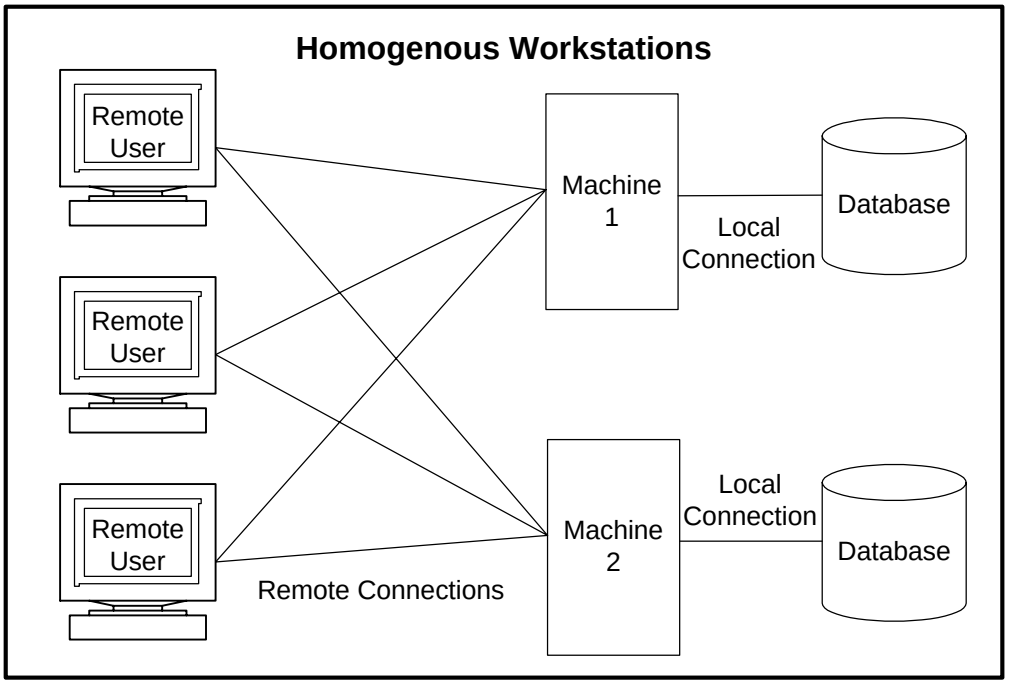


Figure 1–5: Distributed Database Configuration

1.4 Operating System Resources

The Progress Version 9 database relies on operating system resources, such as the file system and processes, to perform its operations. The following sections introduce resources you should be aware of. For more information, see Chapter 14, "Managing Performance."

Semaphores

Semaphores are interprocess communications that act as resource counters. The database engine uses semaphores to synchronize the activities of server and self-service client processes that are connected to a database. By default, each database has an array of semaphores, one for each user or server. Each process uses its semaphore when it must wait for a shared resource. Semaphores are not used for single-user sessions. The Semaphore sets (`-semsets`) Progress broker startup parameter allows you to change the number of semaphore sets available to the broker.

You might also need to set some kernel or system parameters to increase the number of semaphores. For information on setting kernel and system parameters, see the "Semaphores" section of Chapter 14, "Managing Performance."

Spin Locks

The database engine uses a *spin lock algorithm* to control access to shared-memory structures. You can control how the database engine uses spin locks with the Spin Lock Retries (`-spin`) startup parameter. For more information on `-spin`, see the "Unproductive CPU Processing" section in Chapter 14, "Managing Performance."

File Descriptors

A *file descriptor* is an identifier assigned to a file when it is opened. There is a system limit on the number of file descriptors. Each database process (clients, remote client servers, and the broker) uses several file descriptors. You might have to adjust the system file descriptor limit.

Processes

Brokers, servers, clients, and background writers run as processes. Processes implement the client, the server, or the functions for both on a node. Processes also coordinate communication among nodes, and provide operating system and application services for the node. A kernel or system parameter in the operating system limits the number of active processes that can run on a system. You might have to raise the parameter to allow for more processes.

Shared Memory

Shared memory is an area in system memory that multiple users can access concurrently. You might have to adjust the maximum shared-memory segment size and the total number of available segments to improve system performance. For more information on shared memory, see the "Memory Utilization" section in Chapter 14, "Managing Performance."

1.5 Multi-threaded Architecture

The Progress Version 9 database supports a multi-threaded architecture, which provides multiple paths to a database. Each self-service client can access the database and service its own requests. Each network server queues and runs requests for one or more network clients. The database broker initializes shared memory and starts a new server for each additional client or set of clients that access the database. By removing the server architecture bottleneck, multi-threaded architecture increases overall performance.

On multi-processor systems, the multi-threaded architecture allows several processes to access the database simultaneously.

1.6 Multi-tier and Cluster Configurations

For large installations, a multi-tier or cluster configuration might improve system performance. Figure 1–6 illustrates a three-tier configuration that consists of a database-tier that supports self-service clients, an application tier that supports remote clients, and a thin client tier.

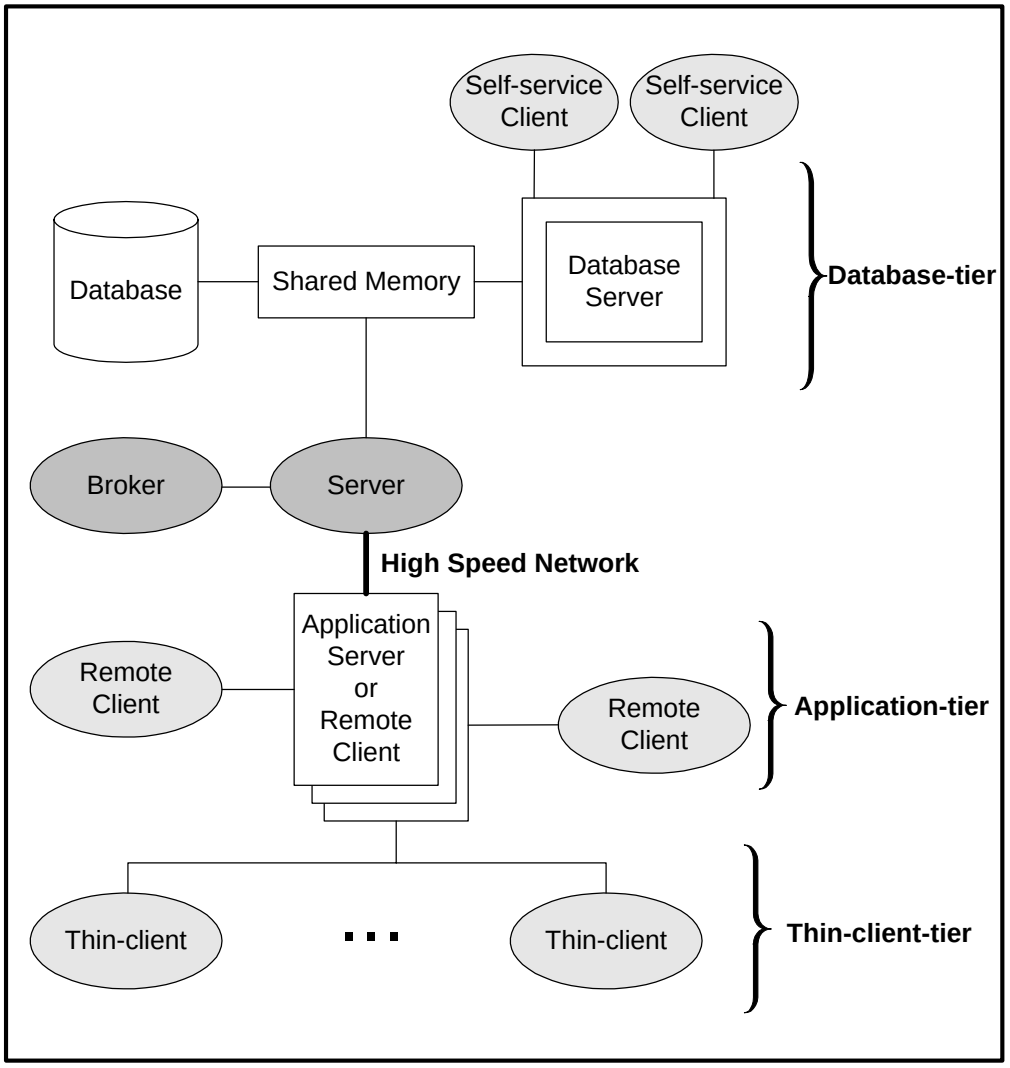


Figure 1–6: Sample Multi-tier Configuration

1.7 Self-service and Network Clients

The Progress Version 9 database architecture supports self-service and network clients. A self-service client is a multi-user session that runs on the same machine as the broker. Self-service clients access the database directly through shared memory and not through servers, because server code is part of the self-service client process. Self-service clients also perform server and client functions in one process, and they execute application logic. On shared-memory UNIX and Windows systems, the database engine provides self-service clients with nearly simultaneous access to the database.

A network client can be local or remote but cannot connect to a database directly; rather it must use a server. The network client accesses the database through a server process that the broker starts over a network connection. The network client does not have access to shared memory, and it must communicate with a server process.

1.8 Relative- and Absolute-path Databases

Progress Version 9 databases can be either relative- or absolute-path. Relative-path databases are only used for special situations, such as installing the master database associated with a new application. Absolute-path databases are used in production environments. The following sections detail relative- and absolute-path databases.

Relative-path Databases

Relative-path databases are the simplest form of Version 9 databases. They contain most of the properties associated with pre-Version 9 (single volume) databases, in that they are made up of a minimum number of files and can be copied with OS commands. A relative-path database stores all extents associated with the database in the same directory as the control area. The control area contains relative paths to all the extents. You would use a relative-path database in the following situations:

- When the database needs to be opened across a network for demonstration purposes
- When the database needs to be copied with OS tools, as when installing a demonstration database
- When the database is the master database to be associated with a new application

Use the PRODB utility to create a relative-path database from an empty database. Figure 1–7 shows the result of using the PRODB utility. The dot at the start of each extent name indicates a relative path.

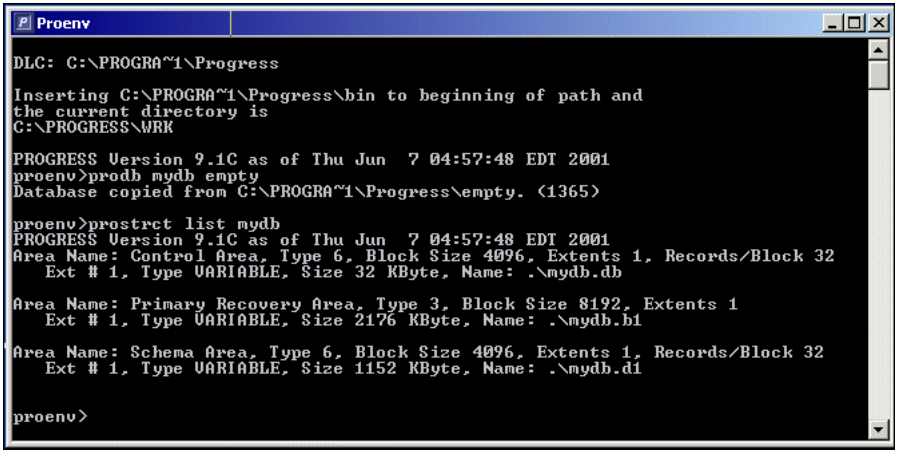


Figure 1–7: Relative-path Database Creation

Schema must be loaded into an empty relative-path database to make it useful. Any standard technique for loading schema, such as a dump and load or by using PROCOPY, can be used. The database maintains its relative-path as long as its structure is not changed. As soon as areas or extents are added, the database becomes an absolute-path database.

Absolute-path Databases

An absolute-path database is used in most production situations. With an absolute-path database, extents associated with the database can be stored anywhere on the system. The control area contains absolute paths to each extent. Absolute-path databases should not be copied using OS tools; rather, PROBKUP and PROCOPY should be used so that all underlying files are properly backed up or copied.

2

Administrative Planning

As the Database Administrator (DBA) of a Progress database, you maintain and administer either an Enterprise, Workgroup, or Personal database. A well-planned Progress database can simplify your job by reducing the time spent maintaining the database structure. This chapter contains the following sections:

- Block Size Considerations
- Calculating Database Disk Requirements
- Storing Database Extents On Raw Partitions

2.1 Block Size Considerations

When planning your database, make sure your block size is equal to, or a multiple of, your file system block size. Matching the database block size to the file system block size will help prevent unneccesary I/O, as shown in Figure 2–1.

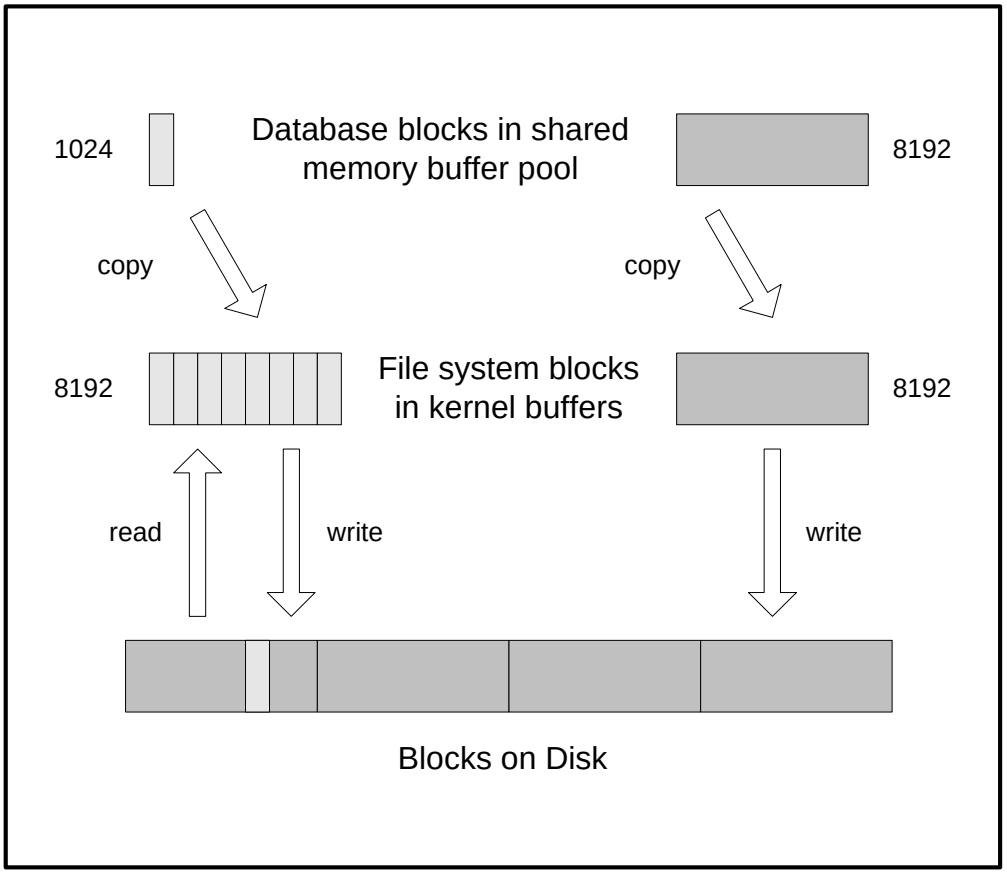


Figure 2–1: Matching Database and File Block Sizes

2.1.1 Guidelines For Choosing Storage Area Locations

When you choose the locations for your database storage areas, consider the following:

- Protect against disk failures by creating the after-image storage area on a separate physical disk from the disks that store the database control and primary recovery areas.

  Never store the primary recovery area extents or data extents on the same physical disk as the after-image extents. Failure of the disk results in the loss of all necessary recovery data.

- Improve performance by creating the primary recovery area on a separate disk from the disk that stores the database control area and its extents.

- If using two-phase commit, create the transaction log area in the same directory as the database control area to simplify management.

2.2 Calculating Database Disk Requirements

The Version 9 database engine stores all database fields and indexes in variable-length format and does not store trailing blanks in character fields or leading zeros in numeric fields. The benefits of this variable-length storage technique are:

- Disk storage is reduced.

- An application is easier to change since you can increase the maximum storage space for a field without copying or restructuring the entire database. You can put more characters into newly created records without affecting existing records.

Table 2–1 lists the formulas you use to calculate the approximate amount of disk space (in bytes) required for a database.

Table 2–1: Formulas For Calculating Database Size *(1 of 2)*

Size	Formula
Database size	*schema size* + *data table size* + *index size*
Schema size	Typically, between 1 and 100MB[1]
Data table size	Sum of individual table sizes

[1] To determine the schema size, load the schema into an empty database and check the size of your database—this is the size of your schema.

Table 2–1: Formulas For Calculating Database Size *(2 of 2)*

Size	Formula
Individual table size	*number of records* x *field storage* x 1.5
Index size	Sum of individual index sizes
Individual index size	*number of records in the table being indexed* x *(7 + number of fields index +field storage)* x *2*

[1] To determine the schema size, load the schema into an empty database and check the size of your database—this is the size of your schema.

Table 2–2 lists the formulas you use to calculate the field storage values (in bytes) for different data types.

Table 2–2: Formulas For Calculating Field Storage *(1 of 2)*

4GL Data Type (SQL-92 Equivalent)	Value	Field Storage In Bytes
CHARACTER (VARCHAR)	1 + *number of characters*, excluding trailing blanks. If the number of characters is greater than 240, add 3 to the number of characters instead of 1.	
DATE (DATE)	3	
DECIMAL (DECIMAL or NUMERIC)	Zero	1
	Nonzero	2 + (number of significant digits + 1) / 2
INTEGER (INTEGER)	Zero to 126	1
	127 to 32,766	2
	32,767 to 7.99 million	3
	8 million to 1.99 billion	4
	2 billion	5

Table 2–2: Formulas For Calculating Field Storage *(2 of 2)*

4GL Data Type (SQL-92 Equivalent)	Value	Field Storage In Bytes
LOGICAL (BIT)	False	1
	True	2

EXAMPLE

You want to calculate the estimated database size for a database with a single customer table. The table has three fields:

- **Cust-num** — An integer field that is always three digits.
- **Name** — A character field containing 12–30 characters, with an average of 21 characters.
- **Start-date** — A date field.

The table is indexed on just one field (Name) and you would expect to have about 500,000 records in the customer table. Table 2–3 lists formulas (and examples) for estimating the size of your database.

Table 2–3: Calculating Database Size *(1 of 2)*

Database Component	Size
Schema	= 421,000 (This number is the size of an empty database with the schema loaded in.)
Field storage	= *Cust-num* + *Name* + *Start-date* = 3 + 21 + 3 = 27
Customer table	= *number of records* x *field storage* x 1.5 = 500,000 x 27 x 1.5 = 20,250,000

Table 2–3: Calculating Database Size *(2 of 2)*

Database Component	Size
Name index	= *number of records* x (7 + *number of fields in index* + *index field storage*) x 2 = 500,000 x (7 + 1 + 21) x 2 = 29,000,000
Database	= *schema size* + *data table size* + *index size* = 421,000 + 20,250,000 + 29,000,000 = 49,671,000 bytes

These formulas are conservative since they often result in a large estimate of your database size. However, you also must allow for temporary disk space for each database user for the following purposes:

- Sorting. Twice the space required to store the table.
- Temporary storage of the primary recovery (BI) area.
- Before-image storage in the primary recovery area.
- After-image storage in the after-image storage area.

Database-related Size Criteria

When planning the size of your database, see Chapter 3, "Progress Database Limits," for a description of the following database-related criteria:

- Database block size
- Number and size of storage areas
- Maximum number of records per block
- Table and index size
- Number of sequences
- Primary recovery area size
- Maximum database size

2.3 Storing Database Extents On Raw Partitions

Progress allows users to store database extents on raw partitions. Storing database extents on raw partitions can sometimes improve performance issues associated with the UNIX file system, such as asynchronous I/O, large file sizes, and security.

To create a database with raw partitions, use the PROSTRCT CREATE command:

```
prostrct create db-name structure-file-name
```

However, the structure file must have the following line format:

```
d    devicename    r    size
```

devicename

Specifies the disk where the database will be created.

size

Specifies the size of the raw partition. You must specify a raw partition size, because a raw partition cannot have a variable-length extent.

NOTE: For the complete syntax of the PROSTRCT utility see Chapter 19, "Database Administration Utilities."

For example, the following structure file creates a database with two data extents and one BI extent on raw partitions. The remaining overflow extents are on the filesystem and look like regular files:

```
d    /dev/rdsk/c0d1s0          r    50000
d    /dev/rdsk/c0d2s0          r    50000
d    /usr1/data/dbname.d1
b    /dev/rdsk/c0d3s0          r    50000
b    /usr2/bifile/dbname.b1
```

NOTE: All databases must have an overflow BI extent that is variable length. This overflow extent must reside on the filesystem.

3

Progress Database Limits

This chapter lists the Progress database limits you need to know when configuring a database and supporting an application development environment. Specifically, this chapter contains the following sections:

- Database Block Sizes
- Number and Size Of Storage Areas
- Maximum Number Of Records Per Block
- Table and Index Limits
- Number Of Sequences
- Maximum Size Of the Primary Recovery (BI) Area
- Maximum Database Size
- Number Of Connections Per Database
- Number Of Simultaneous Transactions Per Database
- Size Of Database Buffer Pool
- Progress Database Name Limits
- Applicable Operating System Limits
- Data Types and Values

3.1 Database Block Sizes

Progress supports the following database block sizes:

- 1024 bytes (1K)
- 2048 bytes (2K)
- 4096 bytes (4K)
- 8192 bytes (8K)

3.2 Number and Size Of Storage Areas

A Progress database supports a maximum of 1,000 storage areas, including 994 application data storage areas. Storage areas are identified by their names. Table 3–1 describes the area types in a Progress database.

Table 3–1: Storage Area Types

Reserved Area Name	Contents	File Extension
Control	Physical Database Structure	.db
Primary Recovery	Recovery Log Data	.b[n]
Transaction Log	Two-phase Commit Transaction Log	.t[n]
After Image	After-image Log Data	.a[n]
Schema	Schema Data	.d[n]
None	Application Data	.d[n]

Application data and schema data storage areas allow you to specify the maximum number of records per block for each area. Table 3–2 describes the maximum application data storage area size determined by database block size and records per block.

Table 3–2: Maximum Application Data Storage Area Size *(1 of 2)*

Database Block Size	Records Per Block	Maximum Area Size
8192 bytes (8K)	1	16TB
8192 bytes (8K)	2	8TB
8192 bytes (8K)	4	4TB
8192 bytes (8K)	8	2TB
8192 bytes (8K)	16	1TB
8192 bytes (8K)	32	512GB
8192 bytes (8K)	64(default)	256GB
8192 bytes (8K)	128	128GB
8192 bytes (8K)	256	64GB
4096 bytes (4K)	1	8TB
4096 bytes (4K)	2	4TB
4096 bytes (4K)	4	2TB
4096 bytes (4K)	8	1TB
4096 bytes (4K)	16	512GB
4096 bytes (4K)	32 (default)	256GB
4096 bytes (4K)	64	128GB
4096 bytes (4K)	128	64GB
4096 bytes (4K)	256	32GB
2048 bytes (2K)	1	4TB
2048 bytes (2K)	2	2TB
2048 bytes (2K)	4	1TB
2048 bytes (2K)	8	512GB

Table 3–2: Maximum Application Data Storage Area Size *(2 of 2)*

Database Block Size	Records Per Block	Maximum Area Size
2048 bytes (2K)	16	256GB
2048 bytes (2K)	32 (default)	128GB
2048 bytes (2K)	64	64GB
2048 bytes (2K)	128	32GB
2048 bytes (2K)	256	16GB
1024 bytes (1K)	1	2TB
1024 bytes (1K)	2	1TB
1024 bytes (1K)	4	512GB
1024 bytes (1K)	8	256GB
1024 bytes (1K)	16	128GB
1024 bytes (1K)	32 (default)	64GB
1024 bytes (1K)	64	32GB
1024 bytes (1K)	128	16GB
1024 bytes (1K)	256	8GB

3.3 Maximum Number Of Records Per Block

You can define the maximum number of records per block for each application data area. When you define an area you can specify 1, 2, 4, 8, 16, 32, 64, 128, or 256 records per block. If you do not explicitly specify the number of records per block when you define an application data area, the default number of records per block is:

- 64 if the block size is 8K
- 32 for all other block sizes

3.4 Table and Index Limits

You cannot split a table or an index across storage areas. Each table and each index can be assigned to only one storage area. Therefore, the size of a table or index is limited to the size of the storage area in which it resides (see Table 3–2).

Regardless of block size, the maximum number of *tables* supported is 32,767.

Regardless of block size, the maximum number of *indexes* supported is 32,767.

Index entries have a maximum of 16 fields per index. Total variable-length storage requirements of all fields in an index entry must be less than 200 characters.

NOTE: Because the 200 character limit includes storage overhead, the actual index key is limited to approximately 188 characters.

3.5 Number Of Sequences

The number of unique sequences supported in a database varies by block size. An application that uses more than 250 sequences does not work on all database block sizes. Table 3–3 lists the number of sequences per database block size.

Table 3–3: Maximum Number Of Sequences

Database Block Size	Maximum Number of Sequences
1024 bytes (1K)	250
2048 bytes (2K)	500
4096 bytes (4K)	1000
8192 bytes (8K)	2000

3.6 Maximum Size Of the Primary Recovery (BI) Area

Only the operating system or the size of your extents imposes a limitation on the size of a primary recovery (BI) area. Multiple BI extents can be added. The maximum possible size of a BI area is 32TB. Table 3–4 lists the maximum BI area size by block size used.

Table 3–4: Maximum Primary Recovery (BI) Area Size

Block Size	Maximum BI Area Size
16,384 bytes (16K)	32TB
8192 bytes (8K) (default)	16TB
4096 bytes (4K)	8TB
2048 bytes (2K)	4TB
1024 bytes (1K)	2TB

3.7 Maximum Database Size

The maximum size of a Progress Version 9 database is determined by the:

- Database block size
- Number of records per block in each area
- Number of storage areas

Database block size and number of records per block determine the maximum storage area size. For example, using one (1) record per block and an 8K (8192) byte block size allows a maximum storage area size equal to 16TB (terabytes).

The result of multiplying the maximum number of areas by the maximum area size identifies the maximum size of a Progress database to be approximately 16PB (petabytes):

1,000 areas (maximum number of areas) * 16TB (maximum area size) = ~16PB.

3.8 Number Of Connections Per Database

Table 3–5 lists the maximum number of connections per database.

Table 3–5: Maximum Number Of Connections Per Database

Database Type	Limit
Single-user	1
Multi-user	Maximum number is machine dependent: up to 10,000 unless constrained by semaphore limits, process limits, or machine performance.

3.9 Number Of Simultaneous Transactions Per Database

Table 3–6 lists the maximum number of simultaneous transactions per database.

Table 3–6: Maximum Number Of Simultaneous Transactions

Database Type	Limit
Single-user	1
Multi-user	1 per user (maximum users = 10,000)

3.10 Size Of Database Buffer Pool

The size of the database buffer pool is set by the `-B` (Blocks in Database Buffers) startup parameter. The maximum is 500,000 blocks. Table 3–7 lists the maximum size of the buffer pool.

Table 3–7: Maximum Buffer Pool Size

Block Size	Maximum Buffer Pool Size
1024 bytes (1K)	500 MB
2048 bytes (2K)	1 GB
4096 bytes (4K)	2 GB
8192 bytes (8K)	4 GB

Progress allows you to specify some number of buffers in the buffer pool as private read-only buffers. The -Bp (Private Read-Only Buffers) startup parameter allows you to specify what number of buffers you wish to be private read-only buffers. The default number of private read-only buffers that a single user can request is 64 and the maximum number is limited by the -bpmax parameter. The maximum number of private read-only buffers for all simultaneous Progress users is limited to 25% of the database buffer pool.

3.11 Progress Database Name Limits

Table 3–8 lists the database name limits for each operating system.

Table 3–8: Database Name Limits

Name Type	Limit
Database names	One to 11 characters, excluding the pathname. You cannot use a file extension.
Pathnames	One to 255 characters, including the database name.

Database names can consist of any combination of English letters and numbers, beginning with A–Z or a–z. They cannot include Progress 4GL or SQL-92 reserved words, any accented letters, or the following special characters:

```
\  "  '  *  ;  |  ?  [  ]  (  )  !  {  }  <  >  @  +  =  :  ~
```

3.12 Applicable Operating System Limits

Progress uses *file handles* (a UNIX term, roughly equivalent to the number of open files) when reading and writing to the Progress database and related files. Most operating systems limit the number of file handles a user process can allocate at one time. Most versions of UNIX guarantee at least 20. Therefore, before you can effectively design a database, you must know your operating system limits and how Progress will use the file handles that are available to it.

Use the following formula to determine the number of file handles Progress uses:

```
H = Static Handles + (# of .dn files) + (# of .bn files) + (# of .an files)
```

H

The number of file handles Progress uses.

Static Handles

The number of handles allocated for any Progress database. The number of static file handles that the process requires depends on whether you are running a Progress client process or a Progress server process:

- Client — Requires nine file handles (PROMSGS + LG + DB + LBI + SRT+ STDIN STDOUT + 2). The file handles used for the input and output devices (STDIN and STDOUT) are allocated by the operating system.
- Server — Requires five handles (PROMSGS + LG + DB + 2).

of .d*n* files

The number of DB files defined for the database.

of .b*n* files

The number of BI files defined for the database.

of .a*n* files

The number of AI files defined for the multi-volume database.

When calculating the number of file handles, consider the following exceptions:

- If you are running a server in a UNIX environment that uses sockets for interprocess communication, add one file handle for each user.
- Application programs use additional file handles when reading and writing text files and when compiling programs. The formula does not account for these additional resource requests.

3.13 Data Types and Values

Table 3–9 lists the Progress SQL-92 data types and value limits.

Table 3–9: SQL-92 Data Type Limits *(1 of 2)*

SQL-92 Data Type	Limit
BIGINT	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
BINARY	2000 bytes
BIT	0 or 1
CHAR	2000 characters
DATE	Year: 1 to 9999; Month: 1 through 12; Date: 1 through the last day of the month
DECIMAL	Defined in terms of precision and scale. Precision=number of digits; scale=number of digits to the right of the decimal point. Note that scale cannot be greater than precision. Precision is limited to 50. Scale is limited to 10.
DOUBLE PRECISION	2.2250738585072014E-308 through 1.7976931348623157E+308
FLOAT	2.2250738585072014E-308 through 1.7976931348623157E+308
INTEGER	-2,147,483,648 to 2,147,483,647
NUMERIC	Defined in terms of precision and scale. Precision=number of digits; scale=number of digits to the right of the decimal point. Note that scale cannot be greater than precision. Precision is limited to 50. Scale is limited to 10.
REAL	1.175494351E-38F to 3.402823466E+38F
SMALLINT	-32,768 to 32,767
TIME	00:00:00 to 23:59:59
TIMESTAMP	Combination of Date and Time limits
TINYINT	-128 to 127

Table 3–9: SQL-92 Data Type Limits *(2 of 2)*

SQL-92 Data Type	Limit
VARBINARY	31,995 bytes
VARCHAR	31,995

The following table lists the 4GL data types and value limits:

4GL Data Type	Limit
CHARACTER	Constrained by record size. (If a field has more than 3,000 bytes, you must write your own dump/reload procedure because the Progress dump/reload procedure cannot handle fields larger than 3,000 bytes.)
DATE	1/1/32768 B.C. to 12/31/32767 A.D.
DECIMAL	50 digits total; 1 to 10 decimal places
INTEGER	–2,147,483,648 to 2,147,483,647.
LOGICAL	TRUE/FALSE, YES/NO.

NOTE: Data columns created using the Progress SQL-92 environment and having a data type that is not supported in the Progress 4GL environment are not accessible by Progress 4GL applications. Data columns created using the Progress 4GL environment can be accessed by Progress SQL-92 applications and utilities.

The following table describes data types supported by Progress 4GL and their corresponding SQL-92 data types:

Progress 4GL Data Type	Progress SQL-92 Data Type
CHARACTER	VARCHAR
DATE	DATE
DECIMAL	DECIMAL or NUMERIC
INTEGER	INTEGER
LOGICAL	BIT
RAW	VARBINARY
RECID	INTEGER

PART II

Administration

Creating and Deleting Databases

Starting Up and Shutting Down

Backup and Recovery Strategies

Backing Up a Database

Recovering a Database

Maintaining Database Structure

Maintaining Security

After-Imaging

Using Two-phase Commit

Dumping and Loading

Managing Performance

Replicating Data

Using the Event Log

4

Creating and Deleting Databases

This chapter describes the methods to create and delete a Progress database.

Specifically, this chapter contains the following sections:

- Ways To Create a Progress Database
- Creating a Database With PROSTRCT CREATE
- Creating a Database With the PRODB Utility
- Creating a Database With the Data Administration Tool
- Migrating Version 8 Databases To Version 9 Databases
- Copying a Database
- AutoConvert Utility
- Deleting a Database

4.1 Ways To Create a Progress Database

There are several ways to create a Progress database:

- From a structure description file with the PROSTRCT CREATE utility
- With the PRODB utility on the command line
- With the Data Dictionary tool if you are using a graphical interface or a character interface
- With the Data Administration tool if you are using a graphical interface
- With the PROCOPY utility
- Convert a Progress Version 8 multi-volume database to a Version 9 database

When you create a database, you can create any of the following:

- A new but empty database
- A copy of an existing database (such as the Progress sports2000 database)

NOTE: Do not create your database in the Progress Install directory ($DLC on UNIX or %DLC% on Windows) or in any subdirectory of the Progress Install directory. Databases residing in these directories cannot be opened.

4.2 Creating a Database With PROSTRCT CREATE

To create a Progress database using PROSTRCT CREATE you must:

- Create a structure description (ST) file to define storage areas and extents.
- Use PROSTRCT CREATE to create a database structure extent.
- Add schema to the void database.

4.2.1 Creating a Structure Description File

The structure description file is a text file you prepare that actually defines the database structure. It contains all of the information required by the PROSTRCT CREATE utility to create a database control area.

Use any text editor, such as vi, edit, or the Progress Procedure Editor, to create the structure description file. The name you give to the structure description file is usually the name of the database you define, with a .st extension.

The structure description file is made up of one or more lines of text that provide information about each storage area of the database. Each line of text is composed of *tokens*, which are text strings made up of alphanumeric characters that describe the following characteristics:

- The storage area type
- The extent pathname
- The extent length
- Optionally, the records per block
- An area name and database engine-generated area number (for application data areas only)

Example Structure Description File

The following example shows the information that is needed in a structure description (ST) file named `sports2000.st` to define a database with:

- One primary recovery area.
- One schema area.
- Three after-image areas each with a fixed-length extent.
- One transaction log area with a fixed-length extent used with two-phase commit.
- Six application data areas each with one fixed- and one variable-length extent. The area names for the six application data areas are: Employee, Inventory, Cust_Data, Cust_Index, Order, and Misc.

```
# Sample Structure Description File: sports2000.st
#
# The following defines the Primary Recovery Area consisting of one
# variable length extent. It resides in the /usr2/bi directory:
#
b /usr2/bi
#
# The following defines the Schema Area consisting of one variable length #
extent, residing in the current working directory:
#
d “Schema Area” .
#
# The following defines three fixed length After Image Areas each equal to
# 1 MB in size:
#
a /usr3/ai f 1024
a /usr3/ai f 1024
a !”/usr3/ai data” f 1024
#
# The following defines a Transaction Log Area equal to 4 MB in size and
# residing in the current working directory. This storage area is used
# for 2 phase commit:
#
t . f 4096
#
# The following defines six Application Data Areas each with one fixed
# length extent equal to 1 MB in size and 1 variable length extent:
#
d “Employee”,32 /usr1/emp f 1024
d “Employee”,32 /usr1/emp
#
d “Inventory”,32 /usr1/inv f 1024
d “Inventory”,32 /usr1/inv
#
d “Cust_Data”,32 /usr1/cust f 1024
d “Cust_Data”,32 /usr1/cust
#
d “Cust_Index”,32 /usr1/cust f 1024
d “Cust_Index”,32 /usr1/cust
#
d “Order”,32 /usr1/ord f 1024
d “Order”,32 /usr1/ord
#
d “Misc”,32 !”/usr1/misc data” f 1024
d “Misc”,32 !”/usr1/misc data”
#
# Note that the directory pathname for the “Misc” application data area
# contains spaces, and to recognize that the pathname is specified with
# an ! (exclamation point) and “ “ (quotation marks).
```

The following is a syntax description of the line format for a structure description (ST) file:

```
CR = blank line
LINE = comment | areatype pathname [ sizeinfo ]
comment = * | : | #
areatype = a | b | d | t [ areainfo ]
areainfo = [“areaname”[:areanum ][, recsPerBlock ] ]
    areaname = string
    areanum = numeric value
    recsPerBlock = numeric value
pathname = . | string | !”string 2"
sizeinfo = extentType size
    extentType = fixed (f) | raw device (r) | variable length (v)
    size = numeric value > 32
```

Note that you can comment the ST file and use blank lines. Precede comments with a pound sign (#), colon (:), or asterisk (*) in the first column of each comment line.

Example Structure Description File For Large Files

When creating a new database, large file processing is enabled if the ST file specifies a fixed length extent size or a maximum size for a variable length extent that is greater than 2 GB. The following example shows the ST file of a database with large file processing enabled.

```
# Sample Structure Description File: largedb.st
#
# largedb.st to create database largedb with large file processing enabled.
#
# A fixed length bi file of 1GB and a variable length bi file with a maximum
# size of 4GB.
#
b tests/largedb.b1 f 1048576
b tests/largedb.b2 v 4194304
#
# SCHEMA AREA with a fixed length file of 3GB and a variable length file with
# a maximum size of 3GB.
#
d "Schema Area":6,64 tests/largedb.d1 f 3145728
d "Schema Area":6,64 tests/largedb.d2 v 3145728
#
# TABLE AREA with a fixed length file of just over 2GB and a variable length
# file with a maximum size of 1TB.
#
d "Table Area":7,64 tests/largedb_7.d1 f 2097280
d "Table Area":7,64 tests/largedb_7.d2
#
# A fixed length ai file of 2GB and a variable length file with a maximum size
# of 1TB.
a tests/largedb.a1 f 2097152
a tests/largedb.a2
#
```

For more information on enabling large file processing, see the "PROUTIL ENABLELARGEFILES Qualifier" section in Chapter 19, "Database Administration Utilities."

Defining Storage Areas and Extents

Define each storage area and extent on a separate line of text in the ST file. Each line is composed of tokens. The first token of each line identifies the type of storage area to be created. These tokens must be lowercase. Table 4–1 lists each token with the corresponding storage area type.

Table 4–1: ST File Tokens and Storage Areas

Token	Type of Storage Area
a	After-image (AI) Area
b	Primary Recovery (BI) Area
d	Schema Area and Application Data Areas
t	Transaction Log (For Two-phase Commit)

Extent Pathnames and Naming Conventions

If the first token defines an application data storage area (d), add an area name after it. Then, the next token defines the file pathname. This pathname must represent a standard operating system file. The only pathname restrictions are those that might be imposed by your operating system's create and read permissions for a particular file.

NOTE: In a structure description (ST) file, to specify a pathname that contains spaces (such as `\usr1\misc data`), precede the pathname with an exclamation point (!) and wrap the pathname in quotation marks (" "). For example, !"`\usr1\misc data`".

The PROSTRCT CREATE utility is designed to allow the end user to specify the minimum amount of information necessary to create a database. Only the area type and extent location must be specified. A specific filename or file extension need not be provided.

The utility will generate filename and file extensions for all database files according to the following naming convention:

- If the pathname is for a raw device, the name is taken as is with no changes.
- If a relative pathname is provided, including using common dot(.) notation, the relative pathname will be expanded to an absolute pathname.
- For BI extents, the filename is the database name with a .b*n* extension, where *n* represents the order in which the extents were created.

- For AI extents, the filename is the database name with a .a*n* extension, where *n* represents the order in which the extents were created.
- For TL extents, the filename is the database name with a .tl extension.
- For Schema area extents, the filename is the database name with a .d*n* extension, where *n* represents the order in which the extents were created and will be used.
- For application data area extents, the filename is the database name and an area number. The area number is a unique identifier set by Progress to differentiate between different areas. The application data area extent filenames also have a .d*n* extension, where *n* represents the order in which the extents were created and will be used.

Rules For Creating Storage Areas and Extents

When you are defining storage areas and extents in order to create a new database:

- The minimum information required in an ST file is one schema area extent definition statement and one primary recovery (BI) area extent definition statement.
- The minimum information needed to specify any extent is the storage area type and extent pathname. For example:

```
# Primary Recovery Area
b .
# Schema Area
d .
```

NOTE: Progress Software Corporation recommends you define a primary recovery area on a different disk than the schema and after-image areas. However, if you do not define a primary recovery extent path in the ST file, the PROSTRCT CREATE utility automatically creates one in the same directory as the database control area.

- You cannot use any of the reserved storage area names defined in Table 4–2 as application data storage area names.

Table 4–2: Reserved Storage Area Names

Reserved Storage Area Name	File Extension Of Extents
Control	.db
Primary Recovery	.b*n*
Schema	.d*n*
After Image	.a*n*
Transaction Log	.t*n*

Extent Length

You can specify a fixed-length or variable-length extent:

- Fixed-length

  When you create a fixed-length extent, its blocks are preallocated and preformatted specifically for the database. If the extent is on a raw partition, the third token of the extent description line is r. If you want the extent to be fixed length, the third token of the extent description line is f. This token must be lowercase. If the extent is fixed length, or raw, use a fourth token to indicate its length in kilobytes.

  The size of the extent, in kilobytes, must be a multiple of 16K. If you specify a size that is not a multiple of 16K, PROSTRCT CREATE displays a warning message and rounds the size up to the next multiple of 16K. The minimum length for a fixed-length file is 16K, and the maximum length of a file depends on the size of the file system and the physical volume containing the extent.

- Variable-length

  Typically, you use a variable-length extent as an overflow file when all fixed-length extents have been exhausted. For DB and BI extents, you can define one variable-length extent for each area. There is no limit to the number of variable-length AI extents you can define.While you indicate a variable-length extent by leaving out the extent size in the ST file entry line, you may also specify the maximum size to which the extent can grow by indicating the “v” extent type and a size in kilobytes. The initial allocation for a variable-length extent is 128K.

4.2.2 Create a Database Structure Extent

Use the PROSTRCT CREATE utility to create the physical database files for the database. The PROSTRCT CREATE syntax is:

```
prostrct create db-name structure-description-file
```

NOTE: For the complete syntax of the PROSTRCT CREATE utility see Chapter 19, "Database Administration Utilities."

For example, to create a database named `sports2000` from the `sports2000.st` structure description file, use PROSTRCT CREATE as follows:

```
prostrct create sports2000 sports2000.st
```

Running the PROSTRCT CREATE utility on `sports2000.st` generates the following extents for a database named `sports2000`:

```
prostrct create sports2000 sports2000.st

Formatting extents:
   size                area name   path name
     32    Primary Recovery Area /usr2/bi/sports2000.b1 00:00:00
     32               Schema Area /usr1/sports2000.d1 00:00:00
   1024       After Image Area 1 /usr3/ai/sports2000.a1 00:00:02
   1024       After Image Area 2 /usr3/ai/sports2000.a2 00:00:03
   1024       After Image Area 3 /usr3/ai/sports2000.a3 00:00:03
   4096    Transaction Log Area /usr1/sports2000.t1 00:00:01
   1024                Employee /usr1/emp/sports2000_7.d1 00:00:01
     32                Employee /usr1/emp/sports2000_7.d2 00:00:00
   1024               Inventory /usr1/inv/sports2000_8.d1 00:00:00
     32               Inventory /usr1/inv/sports2000_8.d2 00:00:00
   1024               Cust_Data /usr1/cust/sports2000_9.d1 00:00:01
     32               Cust_Data /usr1/cust/sports2000_9.d2 00:00:00
   1024              Cust_Index /usr1/cust/sports2000_10.d1 00:00:00
     32              Cust_Index /usr1/cust/sports2000_10.d2 00:00:00
   1024                   Order /usr1/ord/sports2000_11.d1 00:00:01
     32                   Order /usr1/ord/sports2000_11.d2 00:00:00
   1024               Misc /usr1/misc data/sports2000_12.d1 00:00:00
     32               Misc /usr1/misc data/sports2000_12.d2 00:00:01
```

4.2.3 Adding Schema To a Void Database

When you use the PROSTRCT CREATE utility on an ST file, the resulting database files are referred to as a *void* database. A void database does not contain any Progress metaschema information. The void database consists of the DB extent and whatever BI, AI, TL, and D*n* extents you defined in the ST file. You must add Progress metaschema information to a void database. Progress provides an empty database in the size of each supported database block size for this purpose.

NOTE: The empty DB file and the database you want copied to it must have the same block size.

Follow these steps to use the PROCOPY utility to add Progress metaschema information to a void database:

1 ♦ Use the PROCOPY utility to copy the system tables (the Progress metaschema) from a Progress empty database into the void database you created with PROSTRCT CREATE. On UNIX, use the following command syntax:

```
procopy $DLC/emptyn db-name
```

On Windows, use the following command syntax:

```
procopy %DLC%\emptyn db-name
```

2 ♦ Use the PROSTRCT LIST utility to verify that you have the correct database files in the correct locations. The following example uses PROCOPY and PROSTRCT LIST on UNIX:

```
procopy $DLC/empty8 sports

prostrct list sports
```

The following example uses PROCOPY and PROSTRCT LIST on Windows:

```
procopy %DLC%\empty8 sports

prostruct list sports
```

3 ♦ Use the Data Dictionary to load the existing user tables (DF file) into your database.

4.3 Creating a Database With the PRODB Utility

The PRODB utility creates a new database from a specified source database. PRODB creates a new database using the structure of the source database and places all of the extents in the current working directory. You can use PRODB to make a copy of any of the demonstration or empty Progress databases.

NOTE: When using PRODB to create a copy of a database, all the files of the database copy will reside in the same directory, unless you specify a structure description (ST) file for the target database.

EXAMPLES

To create an empty database called `mysample` from a copy of the Progress default empty database, enter the following:

```
prodb mysample empty
```

To create a new database called `mysports2000` from a copy of the Progress sports2000 database, enter the following:

```
prodb mysports2000 sports2000
```

To create a new database called `pastinfo` from a copy of an existing database named `currentinfo`, enter the following:

```
prodb pastinfo currentinfo
```

PRODB does not copy the external triggers associated with the database you are copying.

NOTE: See Chapter 9, "Maintaining Database Structure," for information about changing pathname conventions when adding storage areas and extents to a structure description file.

PRODB Maintains Pathname Convention

When you use the PRODB utility, the target database you create maintains the same pathname convention, relative or absolute, as the source database. For example, if you use PRODB to create a database and name it `example1`, and use a relative path database such as `sports2000` as the source database, PRODB maintains the pathname convention of `sports2000` (the source database), and `example1` (the target database) becomes a relative path database. Conversely, if the source database is an absolute path database, the target database you create with PRODB will also be an absolute path database. Use PROSTRCT LIST to verify whether the pathname is relative or absolute. For example:

```
prodb example1 sports2000
prostrct list example1
```

In the following sample output of the PROSTRCT LIST utility, note the relative pathname of the database, `example1.db`:

```
Area Name: Control Area, Type6, BlockSize 1024, Extents 1, Records/Block32
   Ext # 1, Type VARIABLE, Size 0, Name: ./example1.db

Area Name: Primary Recovery Area, Type3, BlockSize 8192, Extents 1
   Ext # 1, Type VARIABLE, Size 0, Name: ./example1.b1

Area Name: Schema Area, Type6, BlockSize 1024, Extents 1, Records/Block 32
   Ext # 1, Type VARIABLE, Size 0, Name: ./example1.d1

Area Name: Employee, Type 6, BlockSize 1024, Extents 2, Records/Block 32
   Ext # 1, Type FIXED   , Size 320, Name: ./example1_7.d1
   Ext # 2, Type VARIABLE, Size 0, Name: ./example1_7.d2

Area Name: Inventory, Type 6, BlockSize 1024, Extents 2, Records/Block 32
   Ext # 1, Type FIXED   , Size 608, Name: ./example1_8.d1
   Ext # 2, Type VARIABLE, Size 0, Name: ./example1_8.d2

Area Name: Cust_Data, Type 6, BlockSize 1024, Extents 2, Records/Block 32
   Ext # 1, Type FIXED   , Size 320, Name: ./example1_9.d1
   Ext # 2, Type VARIABLE, Size 0, Name: ./example1_9.d2

Area Name: Cust_Index, Type 6, BlockSize 1024, Extents 2, Records/Block 32
   Ext # 1, Type FIXED   , Size 320, Name: ./example1_10.d1
   Ext # 2, Type VARIABLE, Size 0, Name: ./example1_10.d2

Area Name: Order, Type 6, BlockSize 1024, Extents 2, Records/Block 32
   Ext # 1, Type FIXED   , Size 1280, Name: ./example1_11.d1
   Ext # 2, Type VARIABLE, Size 0, Name: ./example1_11.d2
```

4.4 Creating a Database With the Data Administration Tool

Follow these steps to create a database using the Data Administration tool:

1 ♦ Access the Data Administration tool. The Progress Data Administration main window appears.

2 ♦ Choose Database→ Create. The Create Database window appears.

3 ♦ Enter the database name.

4 ♦ Specify whether you want to create a new database, copy the sports database, or copy an existing Progress database.

5 ♦ Specify whether you want to overwrite any existing database with the same name, then choose OK.

If a database already exists with the name you specify, Progress prompts you to verify that you want to overwrite it. If no database with that name exists, the Database Connect dialog box appears.

6 ♦ Enter the appropriate information, then choose OK to return to the Data Administration main window.

4.5 Migrating Version 8 Databases To Version 9 Databases

You can convert Progress Version 8 multi-volume databases to Progress Version 9 databases using the CONV89 qualifier with the PROUTIL utility. If your Progress Version 8 database is single-volume, you must convert it to a multi-volume Version 8 database before using the PROUTIL CONV89 utility. This section describes how to convert Version 8 single-volume databases to Version 8 multi-volume database and how to convert Version 8 multi-volume databases to Progress Version 9 databases.

You can use one of three methods described in this section to convert a Version 8 single-volume database to a Version 8 multi-volume database:

- Use the PROCOPY utility to convert an existing single-volume database to a multi-volume database.
- Use the PROREST utility to convert a single-volume database to a multi-volume database, while minimizing disk space usage.
- Use the PROSTRCT utility to convert an existing single-volume database to a multi-volume database.

For more information about administration utilities, such as PROCOPY, PROREST, PROSTRCT and PROUTIL, see Chapter 19, “Database Administration Utilities.”

4.5.1 Converting a Single-volume Version 8 Database With PROCOPY

Follow these steps to convert an existing Version 8 single-volume database to a multi-volume database:

1 ♦ Create a structure description file to define the appropriate files for the multi-volume database. Give this structure description file a different name from the single-volume database you want to convert.

For example, suppose you have a single-volume database called `finance.db` on a UNIX machine. You might create the following structure description file:

```
*
* This is a structure description file for a multi-volume Progress
* database composed of two fixed-length data extents, one variable-
* length data extent, one BI extent and two variable-length AI
* extents.
*
*
*
*
*
d   /vol1/fsys1/finance2.d1      f  224000
d   /vol2/fsys1/finance2.d2      f   96000
d   /vol2/fsys2/finance2.d3
b   /vol2/fsys2/finance2.b1
a   /vol2/fsys3/finance2.a1
a   /vol2/fsys3/finance2.a2
```

2 ♦ Use the PROSTRCT utility with the CREATE qualifier to create a void multi-volume database structure from the information in the structure description. Give this database the same name as the structure description file. Following is an example of the command:

```
prostrct create finance2
```

3 ♦ Use the PROCOPY utility to copy the single-volume database to the void multi-volume database structure. The source database cannot be in use when you attempt to use PROCOPY. In addition, you cannot use PROCOPY against a crashed database; you must recover the database first. Following is an example of the PROCOPY command:

```
procopy finance finance2
```

The resulting database is then ready to use with Progress applications and can be directly converted to Progress Version 9.

4.5.2 Converting a Single-volume Version 8 Database With PROREST

Follow these steps to convert a Progress Version 8 single-volume database to a multi-volume database, while minimizing disk space usage.

1 ♦ Use the PROBKUP utility to back up the single-volume database. Create at least two backup copies of the database, so that if one copy is damaged, a second copy is available.

2 ♦ Test both backups using the PROREST utility with the Full Verify (`-vf`) parameter.

3 ♦ Delete the single-volume database.

4 ♦ Create a structure description file to define the appropriate files for the multi-volume database. For details, see the "Converting a Single-volume Version 8 Database With PROCOPY" section.

5 ♦ Use PROSTRCT CREATE to create the new void multi-volume structure. For details, see the command description in Chapter 19, "Database Administration Utilities."

6 ♦ Use PROREST to restore the backup into the void structure.

The resulting database is then ready to use with Progress and can be directly converted to Progress Version 9.

4.5.3 Converting a Single-volume Version 8 Database With PROSTRCT

Enter the following syntax, where *db-name* is the name of the database to be converted, to convert a Progress Version 8 single-volume database to a multi-volume database:

```
prostrct convert db-name
```

4.5.4 Converting a Version 8 Database To Version 9

The PROUTIL CONV89 utility converts the schema of a multi-volume Version 8 database to a multi-volume Version 9 database. To convert the schema of a multi-volume Version 8 database to the schema of a Version 9 database, follow these steps:

1 ♦ Using Progress Version 8, back up your database.

CAUTION: There is always a chance that your schema could become corrupt during conversion. The conversion process consists of three phases. If the conversion fails after Phase 1 begins, your database cannot be recovered. Progress issues error messages that your database is corrupt. When this happens, you must revert to the backup copy of your database and begin the conversion again.

2 ♦ Disable after-imaging and two-phase commit.You should disable after-imaging and two-phase commit before starting the conversion; however, if you forget to do so, PROUTIL will disable after-imaging and two-phase commit for you. PROUTIL issues an informational message when after-imaging and/or two-phase commit is disabled.

3 ♦ Truncate your before-image file. PROUTIL will not convert your Version 8 database schema if you do not truncate the before-image file before you start the conversion.

4 ♦ Verify that your Version 8 backup exists, then install Progress Version 9 following the instructions in the *Progress Installation Notes*.

5 ♦ Execute PROUTIL -C CONV89 to convert your database:

```
proutil db-name -C conv89
```

NOTE: For a complete description of the PROUTIL CONV89 utility, see Chapter 19, "Database Administration Utilities."

6 ♦ After you have successfully converted your database, back up your Version 9 database.

You should back up your Version 9 database in case it is damaged due to some failure. If you have only a Version 8 backup of your database, then you would need to go through the conversion process again to convert your Version 8 backup to a Version 9 database. Back up your Version 9 database before you start moving your tables and indexes out of the schema area.

7 ♦ Create at least one application data storage area and move your user tables and indexes into it.

By default, the PROUTIL CONV89 utility places all user tables and indexes in the schema area. Progress Software Corporation recommends that you move them to at least one application data storage area. For instructions on how to create a storage area and move tables and indexes into it, see the "Maintaining Indexes and Tables" section in Chapter 9, "Maintaining Database Structure."

8 ♦ If you intend to enable two-phase commit, add a transaction log area to your Version 9 database. For a description of how to add storage areas to your database see the "Progress Structure Add Utility" section in Chapter 9, "Maintaining Database Structure."

4.6 Using the Schema Mover After Conversion

When you complete the conversion from a Version 8 to a Version 9 database, the database's schema and data are located within the Schema Area (Area 6). After conversion, it is possible to move data into new areas by using PROUTIL dump and load or bulkload qualifiers, the Database Administration Tool, the Database Dictionary, or 4GL code. However, as long as schema remains in an area, the area continues to hold disk space, even after the removal of data. To free this disk space, use PROUTIL with the Schema Mover qualifier to remove the schema. Once the area's schema is removed the area can be truncated.

Follow these steps to move schema to a new area:

1 ♦ Truncate the database's BI file. PROUTIL will send an error message if you do not.

2 ♦ Back up the database.

CAUTION: PROUTIL with the MVSCH qualifier is a non-recoverable utility. If the execution fails, you cannot connect to the database.

3 ♦ Enter the following syntax, where *dbname* is the name of the converted database, to begin the schema move:

SYNTAX

```
proutil dbname -C mvsch
```

As shown in Figure 4–1, after you convert your database from Version 8 to Version 9, Area 6 (NewDB.db) contains both the database's schema and data. When you initiate the schema move by entering the syntax shown in Step 3, PROUTIL finds the next unused data area (starting at Area 7) and creates the target data area and the target data area's extent. After creating the target data area, PROUTIL moves the schema from NewDB.db to NewDB_7.d1. All the schema records and indexes are deleted from NewDB.db. PROUTIL opens the .d1 files of the NewDB.db and NewDB_7.d1 and then updates and switches the master and area blocks. After the switch, the areas and their extents are renamed. Now that the schema is gone from the "Old Default Area," you can truncate it and recover any unused space.

NOTE: Data in the "Old Default Area" not moved prior to area truncation will be lost.

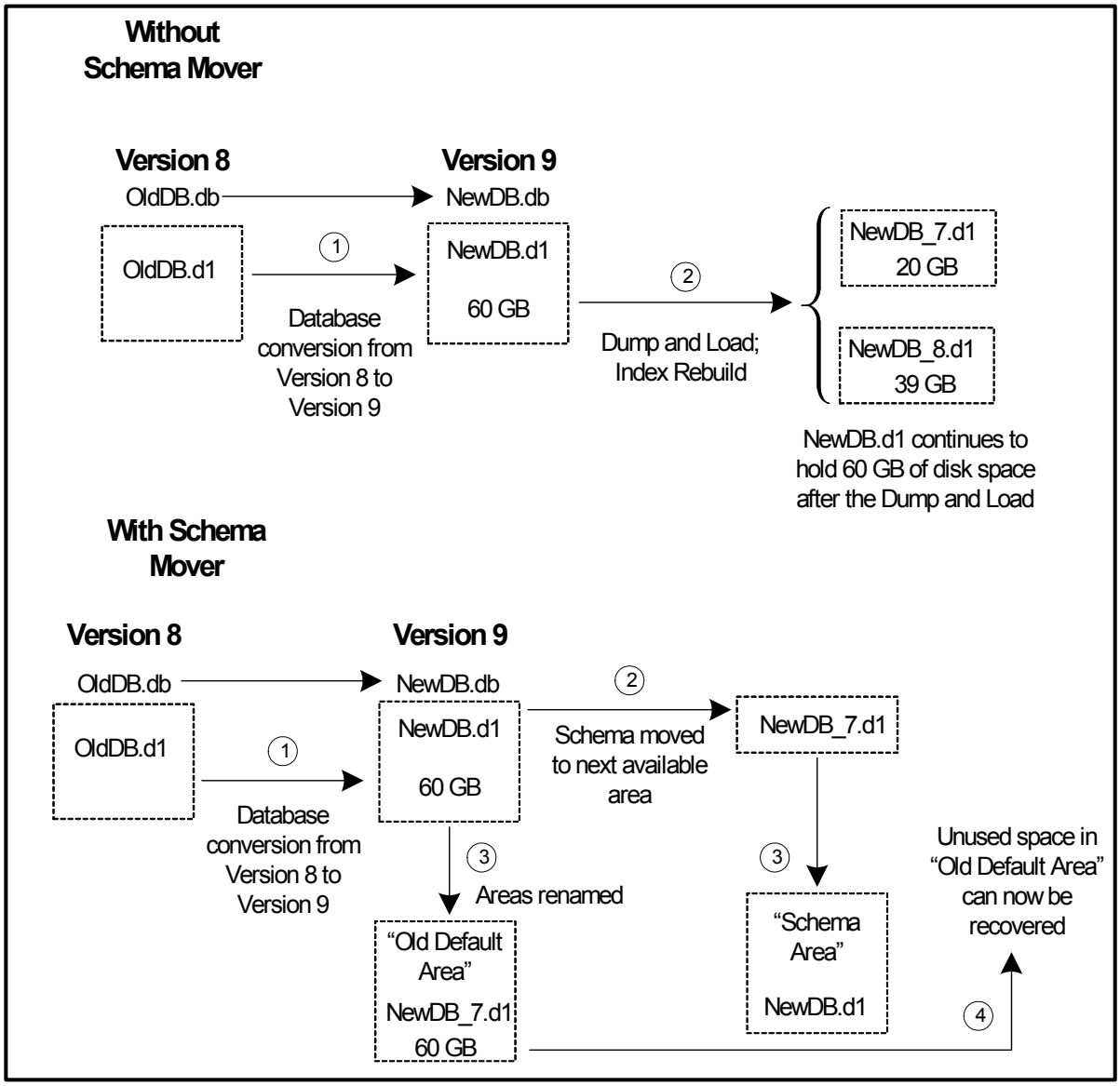


Figure 4–1: How The Schema Mover Works

4.7 Copying a Database

To copy a source database to a target database, use one of the following:

- The PROCOPY utility
- The PRODB utility
- The Progress Data Dictionary

These utilities copy not only the contents of a database but the database structure as well. Consequently, a target database must contain the same physical structure as the source database. For example, it must have the same number of storage areas, records, blocks, and block size.

NOTE: Do not use an operating system utility to copy a Progress database.

For more information about administration utilities, such as PROCOPY and PRODB, see Chapter 19, "Database Administration Utilities."

EXAMPLE

Use the PROCOPY utility to copy an existing database. For example, to copy the `Sports2000` database to a database named `mysports2000`, enter the following:

```
procopy Sports2000 mysports2000
```

PROCOPY supports storage areas. Therefore, if a target database exists, it must contain at a minimum the same type and number of storage areas and same extent types as the source database. However, the number of extents in the storage areas of the target database do not need to match the number of extents in the source database. Progress attempts to extend the existing extents in the target database to accommodate the possible increase in size.

If a target database does not exist, PROCOPY creates one using an existing structure description (ST) file in the target database directory. If an ST file does not exist, PROCOPY creates the target database using the structure of the source database and places all of the extents in the same directory as the target database structure (DB) file, even when the source database resides in multiple directories.

PROCOPY Uses Absolute Pathnames

When you use the PROCOPY utility, the target database you create always has an absolute pathname regardless of the pathname convention used by the source database. For more information on absolute pathnames, see the "Relative- and Absolute-path Databases" section in Chapter 1, "The Progress Database."

For example, if you use PROCOPY to create a database, name it `example2`, and use a relative path database such as `sports2000` as the source database, `example2` will have an absolute pathname even though the source database, `sports2000`, uses a relative pathname. Use PROSTRCT LIST to verify the absolute pathname of your target database. For example:

```
procopy sports2000 example2
prostrct list example2
```

In the following sample output of the PROSTRCT LIST utility, note the absolute pathname of the database, `example2.db`:

```
Area Name: Control Area, Type6, BlockSize 1024, Extents 1, Records/Block32
   Ext # 1, Type VARIABLE, Size 0, Name: /usr1/wrk/example2.db

Area Name: Primary Recovery Area, Type3, BlockSize 8192, Extents 1
   Ext # 1, Type VARIABLE, Size 0, Name: /usr1/wrk/example2.b1

Area Name: Schema Area, Type6, BlockSize 1024, Extents 1, Records/Block 32
   Ext # 1, Type VARIABLE, Size 0, Name: /usr1/wrk/example2.d1

Area Name: Employee, Type6, BlockSize 1024, Extents 2, Records/Block 32
   Ext # 1, Type FIXED   , Size 320, Name: /usr1/wrk/example2_7.d1
   Ext # 2, Type VARIABLE, Size 0, Name: /usr1/wrk/example2_7.d2

Area Name: Inventory, Type6, BlockSize 1024, Extents 2, Records/Block 32
   Ext # 1, Type FIXED   , Size 608, Name: /usr1/wrk/example2_8.d1
   Ext # 2, Type VARIABLE, Size 0, Name: /usr1/wrk/example2_8.d2

Area Name: Cust_Data, Type6, BlockSize 1024, Extents 2, Records/Block 32
   Ext # 1, Type FIXED   , Size 320, Name: /usr1/wrk/example2_9.d1
   Ext # 2, Type VARIABLE, Size 0, Name: /usr1/wrk/example2_9.d2

Area Name: Cust_Index, Type6, BlockSize 1024, Extents 2, Records/Block 32
   Ext # 1, Type FIXED   , Size 320, Name: /usr1/wrk/example2_10.d1
   Ext # 2, Type VARIABLE, Size 0, Name: /usr1/wrk/example2_10.d2

Area Name: Order, Type6, BlockSize 1024, Extents 2, Records/Block 32
   Ext # 1, Type FIXED   , Size 1280, Name: /usr1/wrk/example2_11.d1
   Ext # 2, Type VARIABLE, Size 0, Name: /usr1/wrk/example2_11.d2
```

4.8 AutoConvert Utility

The first time an existing Version 9 database (a database created prior to Progress Version 9.1) is opened for write access, an AutoConvert utility automatically updates the metaschema to support SQL-92 features. The AutoConvert process occurs only once. When it does, the following messages are displayed in the log file:

```
Starting minor version upgrade from 0 to 2. (9111)

Database is now available for updates. (9110)
```

No further action is required.

AutoConvert cannot be executed within a transaction. In order to successfully complete AutoConvert, you must make a direct connection — single-user and not read only — or start a server. If a transaction has already begun when AutoConvert is invoked, the utility detects that transaction, displays an error message, and aborts the session. AutoConvert aborts the session because the database would be in an unsafe state if the outer transaction aborted after another transaction was completed relying on information associated with the upgrade.

NOTE: Once the AutoConvert utility has run, you cannot revert to your previous metaschema. To retain a copy of your pre-AutoConverted metaschema, back up your existing database prior to opening it for write access with Progress Version 9.1.

4.9 Deleting a Database

Use the PRODEL utility to delete a database. For example:

```
prodel mysports2000
```

NOTE: For the complete syntax, see Chapter 19, “Database Administration Utilities.”

When you delete a database, PRODEL displays a message that it is deleting all files that start with *db-name* (the name of the database). PRODEL prompts you to confirm the deletions, depending on your system.

When you delete a database, PRODEL deletes all associated files and extents that were created using the structure description file (database, log, before-image, and after-image files, and with two-phase commit, the transaction log file).

NOTE: The PRODEL utility does not delete the actual structure description file so that a file of your database structure is retained.

5

Starting Up and Shutting Down

This chapter describes how to start up and shut down a Progress database.

Specifically, this chapter contains the following sections:

- The Progress Explorer Framework
- Starting a Server Or Broker
- Starting and Stopping Background Writers
- Stopping a Server Or Broker

5.1 The Progress Explorer Framework

The Progress Explorer Framework is a system administration framework that provides a consistent interface for managing all Progress products installed on your network. The framework consists of the following elements:

- **AdminServer** — Provides secure administrative access to Progress server products.
- **Progress Explorer** — A graphical user interface that provides an easy way for you to manage Progress servers.
- **Command line configuration utilities** — Character versions of the Progress Explorer configuration tools.

5.1.1 AdminServer

An AdminServer is installed on every system where you install a Progress database. The AdminServer grants access to each instance of an installed Progress product. The AdminServer must be running in order to use the Progress Explorer configuration tools or command-line configuration utilities to manage your Progress database.

On Windows NT-based systems, the AdminServer starts automatically and runs as an NT service. For UNIX-based systems, a command-line utility (PROADSV) is used to start and stop the AdminServer. For more information about the AdminServer, see the *Progress Installation and Configuration Guide Version 9 for Windows* or the *Progress Installation and Configuration Guide Version 9 for UNIX*.

5.1.2 Progress Explorer

Progress Explorer is a graphical administration utility that runs on Windows platforms. To use the Progress Explorer configuration tools, you must first start Progress Explorer and connect to a running AdminServer. Progress Explorer then presents you with a view of all the products to which the AdminServer grants you administrative access.

You can select an instance of each of the products displayed and manage its operation or modify its configuration. For example, you can do the following:

- Connect to an AdminServer
- Start, stop, and query the status of Progress databases and associated server groups

For more information about working with the Progress Explorer tool, click the Help icon in the Progress Explorer application.

Starting Progress Explorer

To launch Progress Explorer choose Programs→ Progress→ Progress Explorer Tool from the Windows Start menu.

Managing Database Configurations

The database configurations you create with Progress Explorer are saved in the `conmgr.properties` file. It stores the database, configuration, and server group properties. When you use either the Progress Explorer database configuration tool or the DBMAN utility to start a database, a separate process interprets the information in the properties file and starts and stops whatever server the configuration specifies.

The `conmgr.properties` file resides in the `properties` subdirectory of the Progress install directory.

CAUTION: Do not edit the `conmgr.properties` file directly. Instead, use Progress Explorer to create and edit database configurations.

5.1.3 Command-line Configuration Utilities

The command-line configuration utilities allow you to start, stop, and configure installed Progress products. The database related command-line configuration utilities that are part of the Progress Explorer Framework include:

- **DBMAN** — Starts, stops, and queries the current configuration of a Progress database.
- **PROADSV** — Starts-up and shuts-down an AdminServer on UNIX. For more information about the AdminServer, see the *Progress Installation and Configuration Guide Version 9 for UNIX* or the *Progress Installation and Configuration Guide Version 9 for Windows*.
- **SQLEXP** — Starts SQL Explorer, a tool you use to execute SQL-92 statements. For more information about SQL Explorer, see the *Progress SQL-92 Guide and Reference*.

5.1.4 Using the DBMAN Command-Line Utility

After you use Progress Explorer to create the database configuration and store it in the `conmgr.properties` file, use the DBMAN command-line utility to start, stop, or query a database:

```
dbman [-host host-name -port port-number | service-name -user user-name ]
       -database db-name [-config config-name -start|-stop|-query]
```

The dbman command-line utility supports the following parameters:

`-database` *db-name*

Specifies the name of the database you want to start. It must match the name of a database in the `conmgr.properties` file.

`-config` *config-name*

Specifies the name of the configuration with which you want to start the database.

`-start`

Starts the database *db-name* as defined by the configuration *config-name*.

`-stop`

Stops the database *db-name*.

`-query`

Queries the Connection Manager for the status of the database *db-name*.

`-host` *host-name*

Identifies the host machine where the AdminServer is running. The default is the local host. If your AdminServer is running on a remote host, you must use the `-host` *host-name* parameter to identify the host where the remote AdminServer is running.

`-port` *port-number*|*service-name*

Identifies the port that the AdminServer is listening on. If your AdminServer is running on a remote host, you must use the `-port` *port-number* parameter to identify the port on which the remote AdminServer is listening. The default port number is 20931.

`-user` *user-name*

If your AdminServer is running on a remote host, you must use the `-user` *user-name* parameter to supply a valid user name for that host. You will be prompted for the password.

5.2 Starting a Server Or Broker

The server process coordinates all the database requests from all the users using a single database.

NOTE: On UNIX and Windows NT systems that run shared-memory versions of Progress, the main database server is called the *broker*. The broker process manages shared resources and starts servers for remote users, if necessary. For more information, see the "Operating System Resources" section in Chapter 1, "The Progress Database."

5.2.1 Using the PROSERVE Command

Use Progress Explorer or use the PROSERVE startup command to start the server process:

```
proserve -db db-name | -servergroup servergroup-name [ parameters ]
```

`-db` *db-name*

Specifies the database you want to start Progress against. (`-db` is implicit)

`-servergroup` *servergroup-name*

Specifies the logical collection of server processes to start. The *servergroup-name* you specify must match the name of a servergroup in the `conmgr.properties` file. You create servergroups using the Progress Explorer Database Configuration Tools, which saves them in the `conmgr.properties` file.

parameters

Specifies the startup parameters for the broker/server. See Chapter 18, "Database Startup Parameters," for a list of broker/server startup parameters.

For more information about the PROSERVE command see Chapter 17, "Startup and Shutdown Commands."

5.2.2 Specifying International Character Sets

A Progress international database has one character set (code page) associated with all its data. This *database character set* information is stored in the database.

In addition, a database server has an *operating character set*. The operating character set is used for every character operation such as, compare and substring. You can use the Internal Code Page (`-cpinternal`) international startup parameter, to define the operating character set. If you do not use `-cpinternal`, the default operating character set is iso8859–1.

Specifying the Operating Character Set

Use Progress Explorer to look up the operating character set name in the database configuration properties windowpane and reset it appropriately.

To specify the operating character set from the command line, use the PROSERVE administration utility. For example:

```
proserve db-name -cpinternal character-set-name
```

Specifying the Character Set Name Of the Database Log File

To specify the output character set for database log messages, use Progress Explorer to look up the log character set (LogCharacterSet) name in the database configuration properties windowpane and reset it appropriately.

Otherwise, use the Log Character Set (`-cplog`) international startup parameter with the PROSERVE administration utility. For example:

```
proserve db-name -cplog character-set-name
```

For more information on character sets and character conversion, see the "PROUTIL Utility" section in Chapter 19, "Database Administration Utilities."

5.2.3 Network Addressing With Progress (- S and -H)

In all network environments, you use the Service Name (-S) startup parameter to assign a name to a Progress broker/server. You then address this broker/server from a remote client by using the same value for -S as a startup or database connection parameter. Depending on your network type, you might also have to specify additional addressing criteria for remote Progress clients. In terms of Progress addressing, the TCP protocol uses host addressing.

The TCP protocol requires a remote client to explicitly address the database server machine (or host) on which the server runs. In a TCP network, you must use the Host Name (-H) startup parameter to specify the host address. The -H value is the name assigned to the database server machine in your TCP/IP hosts file.

Use Host Name (-H) to identify the host name. For example:

-H *host-name*

The TCP/IP host-name (address) of the database server machine.

-H localhost

A reserved word that specifies that the Database server communicates only with clients on the database server machine. Not applicable for DataServers.

NOTE: The -H parameter has a special purpose when used with the Progress/400 DataServer and SNA. If you are connecting to multiple AS/400 machines, you must supply this parameter for each AS/400.

5.2.4 Starting Network Brokers and Servers

You can start most network brokers and servers using the PROSERVE command for your database server machine. For example, on a UNIX machine, you enter the following command to start brokers for two databases (sports and news) using the TCP network type:

```
proserve sports -S spsrv -N TCP -db news -S nwsrv -N TCP
```

5.2.5 Starting Multiple Brokers Using the Same Protocol

You can start multiple brokers that use the same protocol. The -Mn parameter and a new parameter, Maximum Servers per Broker (-Mpb), determine the number of servers a broker can start. In addition, you can use Progress Explorer to manage and configure servergroups.

Use the following commands to start two brokers that use TCP and start multiple servers each:

```
proserve db-name -S server-name -H host-name -Mnn -Mpbn
proserve db-name -S server-name -H host-name -Mpbn -m3
```

db-name

Specifies the database you want to start. If the database is not in the current directory, you must specify the full pathname of the database.

-S *service-name*

Specifies the database server or broker process service name. You must specify the service name in a TCP network.

-H *host-name*

Specifies the machine where the database server runs.

-Mn *n*

Specifies the maximum number of remote client servers and login brokers that the broker process can start.

-Mpb *n*

Specifies the number of servers that the login broker can start to serve remote users. This applies to the login broker that is being started.

-m3

Starts the secondary login broker.

For example, you would use the following commands to start two brokers that use TCP and start four servers each:

```
proserve db -S demosv1 -H myhost -Mn 9 -Mpb 4
proserve db -S demosv2 -H myhost -Mpb 4 -m3
```

As the example shows, the -Mn value must be large enough to account for each additional broker and all servers. If you do not specify -Mpb, the value of -Mn becomes the default.

You must include the -m3 parameter with every secondary broker startup command. While the -Mpb parameter sets the number of servers a broker can start, the -m3 parameter actually starts the secondary broker.

If you start multiple brokers, you should also run the Progress Watchdog process (PROWDOG). PROWDOG enables you to restart a dead secondary broker without shutting down the database server. For more information on PROWDOG, see "PROWDOG Command" section in Chapter 17, "Startup and Shutdown Commands."

5.2.6 Accessing a Server Behind a Firewall

Progress allows you to use the Minimum Dynamic Server Port (-minport) and the Maximum Dynamic Server Port (-maxport) server startup parameters to provide client access to a Progress server that is behind a firewall. This communication is possible only when the access to the server can be limited. You supply this limit when you specify a group of port numbers with the -minport and -maxport parameters.

For example, suppose you start the following two login brokers:

```
proserve db -S demosv1 -H myhost -minport 4000 -maxport 4040
proserve db -S demosv2 -H myhost -minport 4041 -maxport 4080 -m3
```

A client requesting a connection from the first broker, demosv1, is assigned a port number in the range of 4000 to 4040. The 4000-to-4040 range limits access to the server by limiting communication to just 41 ports.

The default for -minport is 1025 for all platforms. Ports lower than 1025 are usually reserved for system TCP and UDP. The default for -maxport is 2000 for all platforms. Remember that some operating systems choose transient client ports in the 32,768-to-65,535 range. Choosing a port in this range might produce unwanted results.

5.3 Starting and Stopping Background Writers

On shared-memory systems, background writers improve performance by continually performing overhead functions in the background. You must manually start these background writers.

There are three types of background writers: asynchronous page writers (APWs), before-image writers (BIWs), and after-image writers (AIWs). The following sections explain how to start up and shut down background writers. See Chapter 14, "Managing Performance," for detailed information about background writers.

5.3.1 Starting and Stopping an APW

A database can have between zero and nine APWs running simultaneously. The optimal number is highly dependent on your application and environment. To start, use one page writer for each disk where the database resides. If this is insufficient, add more. For an application that performs many updates, start one APW for each disk containing your database, plus one additional APW. Applications that perform fewer changes to a database require fewer APWs.

NOTE: If you perform no updates, no page writers are required.

To start an APW process, use Progress Explorer or enter the following command on the local machine:

```
proapw db-name
```

Each APW counts as a process connected to a database and uses resources associated with a user. You might have to increase the value of the Number of Users (-n) parameter to allow for APWs. However, APWs are not counted as licensed users.

Stop an APW by disconnecting the process with the PROSHUT command.

For detailed information on the PROAPW and PROSHUT commands, see Chapter 17, "Startup and Shutdown Commands."

5.3.2 Starting and Stopping a BIW

You can only run one BIW per database. You can start and stop the BIW process at any time without shutting down the database.

To start the BIW process, use Progress Explorer or enter the following command on the local machine:

```
probiw db-name
```

The BIW counts as a process connected to a database and uses resources associated with a user. You might have to increase the value of the Number of Users (-n) parameter to allow for the BIW. However, the BIW is not counted as a licensed user.

Stop the BIW by disconnecting the process with the PROSHUT command.

5.3.3 Starting and Stopping an AIW

You can run only one AIW process per database at a time. You can start and stop an AIW at any time without shutting down the database.

To start the AIW process, use Progress Explorer or enter the following command:

```
proaiw db-name
```

You must have after-imaging enabled to use an AIW. For more information on after-imaging, see Chapter 11, “After-Imaging.”

The AIW counts as a process connected to a database and uses resources associated with a user. You might have to increase the value of the Number of Users (-n) parameter to allow for the AIW. However, the AIW is not counted as a licensed user.

To stop the AIW process, disconnect it with the PROSHUT command.

5.4 Stopping a Server Or Broker

Before you turn off your computer or back up the database, you must shut down the server or broker process. Before you shut down the server, all application users must quit their Progress sessions. If necessary, you can disconnect all users with the PROSHUT command's Disconnect a User or Unconditional Shutdown qualifiers. For more information, see the "PROSHUT Command" section.

You can shut down a database using:

- Progress Explorer
- DBMAN utility
- PROSHUT command
- PROMON utility

To shut down the database, you must either be the user who started the database or have root privileges.

NOTE: Do not use operating system commands to shut down the database.

5.4.1 PROSHUT Command

To shut down a Progress database server with the PROSHUT command, enter one of the following:

```
proshut db-name [ -b | -by | -bn | -H host-name | -S service-name
                -F | -Gw
```

db-name

Specifies the database the server is running against.

-b

Indicates a batch shutdown will be performed. When no client is connected, the database automatically shuts down. When one or more clients are connected, PROSHUT prompts the user to enter "yes" to perform an unconditional batch shutdown and to disconnect all active users; or "no" to perform a batch shutdown only if there are no active users. The -b parameter combines the functionality of the -by or -bn parameters.

-by

Directs the broker to perform an unconditional batch shutdown and to disconnect all active users.

-bn

Directs the broker to perform a batch shutdown only if there are no active users.

-H *host-name*

Specifies the machine where the database server runs. You must specify the host name if you issue the shutdown command from a machine other than the host.

-S *service-name*

Specifies the database server or broker process service name. A TCP network requires the -S parameter.

-F

Forces an emergency shutdown. To use this parameter, you must run PROSHUT on the machine where the server resides, and on UNIX systems. This parameter is not applicable for remote shutdowns or DataServer shutdowns.

CAUTION: Using -by with -F causes an emergency shutdown.

-Gw

For DataServers, specifies the DataServer broker to shut down.

For complete PROSHUT syntax, see the "PROSHUT Command" section in Chapter 17, "Startup and Shutdown Commands."

When you enter the PROSHUT command without the -by, -bn, or -F parameters, the PROSHUT menu appears:

```
                        1 Disconnect a User
                        2 Unconditional Shutdown
                        3 Emergency Shutdown (Kill All)
                        x Exit
```

Table 5–1 lists the PROSHUT menu options and their actions.

Table 5–1: PROSHUT Menu Options

Option	Action
1	Prompts you for the number of the user you want to disconnect.
2	Stops all users and shuts down the database. If you have a shared-memory system with multiple users, Progress stops all servers. To stop a specific server process, use the appropriate operating system command.
3	Prompts you to confirm your choice. If you cancel the choice, you cancel the shutdown. If you confirm the choice, PROSHUT waits for five seconds before taking any action, then displays the following message: `Emergency shutdown initiated...` PROSHUT marks the database for abnormal shutdown and signals all processes to exit. After 10 more seconds, PROSHUT kills all remaining processes connected to the database, and deletes shared-memory segments and semaphores. The database is in a crashed state. Progress performs normal crash recovery when you restart the database and backs out any active transactions. This option is available only if the database is on the same machine where you are logged in.
x	Cancels the shutdown without taking any action.

If you want to execute the shutdown command noninteractively and avoid the PROSHUT menu, issue the PROSHUT command using either of the parameters described in Table 5–2.

Table 5–2: PROSHUT Parameters

Parameter	Action
Kill Users (`-by`)	Unconditional batch shutdown; kills all active users.
Proceed If No Users (`-bn`)	Batch shutdown only if there are no active users.

In a TCP/IP network, when using the shutdown command from a machine other than the host, you must use the Host Name (-H) and Service Name (-S) parameters. The Host Name is the machine where the database server is running. The Service Name is the name of the database server or broker process, as defined in the /etc/services file on UNIX. For example, the following command shuts down the sports database from a remote machine in a BSD UNIX network:

```
proshut sports -H host-name -S sports-broker -by
```

5.4.2 PROMON Shut Down Database Option

You can use the PROMON utility to stop the database or disconnect any subset of users.

Follow these steps to shut down a database using PROMON:

1 ♦ Enter the following PROMON command:

```
promon db-name
```

When you start the monitor, the Database Monitor main menu appears:

```
Progress MONITOR
Database: PROGRESS/WRK/sports2000
1.  User Control
2.  Locking and Waiting Statistics
3.  Block Access
4.  Record Locking Table
5.  Activity
6.  Shared Resources
7.  Database Status
8.  Shut Down Database

T.  Transaction Control
L.  Resolve Limbo Transactions
C.  Coordinator Information

M   Modify Defaults
Q.  Quit

Enter your selection:
```

2 ♦ Choose option 8, Shut Down Database.

The following figure shows an example of this option's output:

```
Enter your selection:  8

Usr    PID      Time of login     Userid  tty             Limbo?
1      6358     Dec 14 15:10:52   sue     /dev/ttyp0      no
4      7007     Dec 14 15:25:09   mary    /dev/ttyp5      no

               1 Disconnect a User
               2 Unconditional Shutdown
               3 Emergency Shutdown (Kill All)
               4 Exit

Enter choice>
```

3 ♦ Choose an option.

If you choose 1 (Disconnect a User), the system prompts you for a user number. Choose 2 (Unconditional Shutdown) to stop all users and shut down the database. If you have multiple remote-user servers, this stops all the servers.

6

Backup and Recovery Strategies

Backup and recovery strategies work together to restore a database that is lost due to hardware or software failure. It is important to develop backup and recovery strategies that you can follow consistently and effectively. This chapter lists the steps needed to develop effective backup and recovery plans. Specifically, it contains the following sections:

- Identifying Files For Back Up
- Determining the Type Of Backup
- Choosing Backup Media
- Creating a Backup Schedule
- Developing a Recovery Plan
- Recovery Guidelines

6.1 Identifying Files For Back Up

To correctly back up a Progress database, you must archive all of the files associated with the database. Files you must back up are:

- Database files (.db, .b*n*, d*n*)

  These files contain data and recent transaction information. You must back up these files as a unit; you need all these files to recover a consistent database.

- After-image files (.a*n*)

  After-image (AI) files contain information required to reconstruct a database if a database disk is lost or damaged. You roll forward these files to reprocess all transactions that occurred since the last backup. Archive each AI file when it is full or when the database is not in use. You must use an operating system backup utility.

- Event log files (.lg)

  Event log (LG) files contain dates and times of important database events. They also contain messages and other historical information to help you diagnose complex problems by understanding the circumstances surrounding a failure. Back up these files regularly as part of the normal system backup procedures. You must use an operating system backup utility.

- Transaction log files (.tl)

  If you use two-phase commit, the transaction log files contain information used to resolve in-doubt two-phase commit transactions. Back up these files regularly as part of the database backup procedures. You must use an operating system backup utility.

- Application files and program library files (.pl)

  Back up these files regularly as part of database backup procedures. You must use an operating system backup utility.

  If you store database triggers in a program library, you should maintain the library to make sure it matches the database schema. When you back up the database, PROBKUP backs up the database schema and any schema references to 4GL triggers, but it does not back up 4GL trigger code that is referenced in the schema.

6.2 Determining the Type Of Backup

It is possible to use either a Progress backup (the PROBKUP utility) or a non-Progress (Operating System) backup. Which you choose depends upon the needs of your business. Table 6–1 contrasts Progress with non-Progress backups.

Table 6–1: Backup Options

Progress	Non-Progress
Online or Offline	Offline Only
Full or Incremental	Full Only

To determine whether an online or offline backup should be used, ask:

- Is the database active 24 hours a day, 7 days a week? Is it possible to shut down the database for backing up?

  If the database must run 24 hours a day, 7 days a week, an online backup is necessary. If it does not, an offline backup can be used.

To determine whether a full or incremental backup is best, ask:

- Does the entire database fit on one volume of backup media? If not, will someone be present to change volumes during the backup?

  If the database fits on one volume or someone is present to change volumes, a full backup can be performed. If not, consider using incremental backups.

Progress Software recommends that you use the Progress Backup (PROBKUP) utility to perform database backups for the following reasons:

- PROBKUP automatically backs up all files required to recover a consistent database.
- PROBKUP allows both online and incremental backups, in addition to offline and full backups.
- PROBKUP allows users access to the database during an online backup.
- PROBKUP automatically marks the database as backed up.
- The Progress Restore (PROREST) utility lets you easily verify backups.

If you choose not to use PROBKUP, you can use an operating system backup utility, but you cannot perform online or incremental backups. Table 6–2 lists the backup utilities for various operating systems.

Table 6–2: Operating System Backup Utilities

Operating System	Backup Utilities
UNIX	`tar`, `cpio`, or a manufacturer-supplied backup utility
Windows	Windows backup or any backup utility that can back up and restore individual files

CAUTION: Backups performed by an operating system utility must be done offline, unless the PROQUIET utility is used with disk mirroring. Be sure that the backup utility you choose backs up the entire set of files. Backing up a partial database provides an invalid result.

NOTE: If you do not use the Progress backup utility and chose to use another mechanism, Progress Software Corporation recommends that you take great care to ensure that the other mechanism provides a valid backup. Regardless of the backup method chosen, perform a complete database restore using PROREST to test all backups and validate that the database is correct.

A full backup backs up all of the data of a database, including the BI files. You can perform a full backup using either the PROBKUP utility or an operating system utility. Ideally, you should perform a full backup of your database every day. However, depending on your recovery plan, you might decide to do less frequent full backups and more frequent incremental backups, or use after-imaging.

6.2.1 Incremental Backups

An incremental backup backs up only the data that has changed since the last full or incremental backup. Incremental backups might take less time and media to back up the database; the amount of time you can save depends on the speed of your backup device. You must use PROBKUP to perform an incremental backup.

In a Progress database, the master block and every database block contains a backup counter. The counter in the master block is incremented each time the database is backed up with an online, offline, full, or incremental backup. When a database block is modified, Progress copies the backup counter in the master block to the backup counter in the modified database block. When you perform an incremental backup, PROBKUP backs up every database block where the counter is greater than or equal to the master block counter. If you specified the Overlap (-io) parameter, PROBKUP backs up every database block where the counter is greater than or equal to the master block counter, less the overlap value you specify. For more information on the -io parameter, see the "Performing a Progress Backup" section in Chapter 7, "Backing Up a Database."

You must perform a full backup of a database before you can perform the first incremental backup. You should also perform full backups regularly in conjunction with incremental backups. If you do not perform full backups regularly, you will use large amounts of backup media and increase recovery time.

6.2.2 Online Backups

An online backup lets you back up the database while it is in use. You must use PROBKUP to perform online backups. Perform an online backup if the database cannot be taken offline long enough to perform an offline backup. You can perform both full and incremental online backups.

When deciding whether to use online backups, consider the following:

- You can perform an online backup if after-imaging is enabled. See Chapter 11, "After-Imaging," for more information about after-imaging.
- You cannot perform an online backup on a system running in single-user mode.
- When you perform an online backup of a database, the database engine automatically switches over to the next AI file. Before you start an online backup, you must make sure that the next AI file is empty. If the file is not empty, PROBKUP aborts and notifies you that it cannot switch to the new file. See Chapter 11, "After-Imaging," for information about the AIMAGE EXTENT LIST qualifier with the RFUTIL utility.
- The PROQUIET utility allows you to put the database in a quiet state while users are connected. You can then use backup techniques without shutting down the database. However, no active use of the database is allowed while the database is quiet. This approach is useful when split-mirrors are used as a backup technique.

- When you begin an online backup, make sure that you properly mount the backup media. If the media is not properly mounted, the database engine cannot write header information to it. Until the database engine writes the header information, you cannot update the database. If you use more than one volume to back up the database, there is a similar delay each time you switch volumes.
- You cannot use the PROBKUP parameters Scan (`-scan`) or Estimate (`-estimate`) for online backups.

6.2.3 Offline Backups

To perform offline backups, you must first shut down the database. If you perform an offline backup with an operating system utility on a database while it is running, the backup is invalid. You can use PROBKUP or an operating system utility to perform offline backups.

For more information, see Chapter 8, "Recovering a Database."

6.3 Choosing Backup Media

Depending on the operating system, there are many different options available for backup media. Media available for UNIX and Windows include:

- Disk files (single or multiple)
- Tape cartridges
- Disk cartridges
- Writable compact disks

When choosing backup media, consider the media's speed, accessibility, location, and capacity for storage. Table 6–3 lists questions to ask yourself when choosing backup media.

Table 6–3: Backup Media Questions *(1 of 2)*

To Consider This. . .	Ask This. . .
Storage capacity	Is the media large enough to contain all the files you need to back up?
Data transfer speed	Is the media fast enough to allow backup of files in a reasonable time?

Table 6–3: Backup Media Questions *(2 of 2)*

To Consider This. . .	Ask This. . .
Accessibility	Can you use the media at the time you need to run the backup?
Location	Is the device accessed through a network? If so, can the network handle the large volume of activity generated during a backup?

6.4 Creating a Backup Schedule

The backup schedule is a fundamental part of the recovery plan. It determines how frequently you must perform backups, assigns responsibility for making backups, and serves as a record of when backups are performed.

Ideally, you should perform a full backup of the database every day. However, depending on availability requirements and your recovery plan, a less frequent full backup and frequent incremental backups might be sufficient. For information on determining availability requirements and developing a recovery plan, see Chapter 8, "Recovering a Database."

Several considerations affect the backup schedule:

- Database integrity
- Database size
- Time
- Unscheduled backups

6.4.1 Database Integrity

To preserve database integrity:

- Back up the database frequently.
- Back up the AI files separately from the database backup.
- Use the PROREST verify parameters Partial Verify (`-vp`) and Full Verify (`-vf`) to verify that a backup is valid.

6.4.2 Database Size

If the database is very large, it might be impractical to fully back up the database daily. You might choose to back up the database file every other day, or even once a week. Instead of performing daily full backups, consider performing daily incremental backups or, if you have after-imaging enabled, only backing up the after-image file.

You can perform daily incremental backups. Incremental backups only back up the blocks that have changed since the previous backup. You can specify an overlap factor to build redundancy into each backup and help protect the database. However, you should also perform a full backup at least once a week to limit the amount of backup media used for incremental backups and to ease data recovery. See Chapter 7, "Backing Up a Database," for more information about incremental backups.

If you enable after-imaging, back up the after-image files every time you perform a backup. Immediately after performing a full or incremental backup, start a new after-image file. The after-image file continues to grow until the next time you back up the database and after-image file. When you back up AI files, you back up whole transactions, but incremental backups back up just the blocks that have changed. As a result, AI file backups can use more space than incremental backups.

If you make many database updates and you are on a weekly full backup schedule, it is possible that the after-image file will grow very large during the week. If so, backup the AI file and start a new one every day. This daily backup approach keeps the AI file relatively small and ensures that the AI file is archived on a regular schedule.

NOTE: PROBKUP does not back up AI files. You must use an operating system backup utility.

6.4.3 Time

When creating a backup schedule, consider both the time required to perform backups and the time required to recover a database from the backups:

- Performing daily full backups of your system might require too much time. If you make few updates to the database each day, a daily full backup might be unnecessary. Thus, you might want to perform daily incremental backups and a weekly full backup. If you have after-imaging enabled, remember to back up the after-image files for both incremental and full backups.

- If you perform incremental backups less frequently than once a day, you risk losing substantial amounts of data. If you perform full backups less than once a week, you must maintain multiple incremental backups, which makes recovery more complicated and prone to operator error.

- If you enable after-imaging, you can perform daily backups of the after-image file instead of performing incremental backups. However, recovering from the AI file backups requires restoring the AI files then rolling forward through multiple after-image files. Because backing up AI files backs up whole transactions instead of just the blocks that have changed since the most recent backup, restoring a database from incremental backups is quicker than restoring AI files and rolling forward the AI files.

6.4.4 Unscheduled Backups

In addition to scheduled backups, you might have to perform additional backups for the following reasons:

- To run a large job with the No Crash Protection (`-i`) startup parameter. Before running the job, back up the database and after-image file.
- To re-establish a valid environment required by the recovery plan.
- As part of the recovery process.
- Before and after any major changes to an application or database.
- When installing a new version of Progress.

6.5 Developing a Recovery Plan

A recovery plan documents how you will recover your system if it fails. When developing a recovery plan, you must consider every potential failure, from short-term power outages and disk failures to environmental disasters such as earthquakes.

To develop a database recovery plan, determine the availability requirements for the database application. Consider the following questions:

- How many transactions can you afford to lose?
- How long can the application be offline while you perform scheduled maintenance, such as backups?
- If the system or database becomes unavailable, how much time can you spend recovering?
- If you use transactions that affect more than one database, can you allow transactions to occur inconsistently in those databases?
- How will you test your recovery plan?

Use the tables in the following sections to develop a recovery plan. These tables provide a range of answers to these questions and backup and recovery suggestions for each.

6.5.1 Lost Transactions

The type of recovery mechanisms you use depends on the number of committed transactions you can afford to lose if the database becomes unavailable. Table 6–4 provides examples.

Table 6–4: Availability and Transaction Loss

If You Can Lose . . .	Then Use . . .
No committed transactions	After-imaging
One day of committed transactions	Daily backups
One week of committed transactions	Weekly backups

6.5.2 Duration Of Offline Applications

The type of backup you perform depends on how long you can take the database offline while you back it up. Table 6–5 provides examples.

Table 6–5: Availability and Backup Time

If the Application Can Be Offline . . .	Then Perform . . .
Never	Online backups with after-image files.[1]
Eight hours per week	Weekly offline backups with daily online full or incremental backups. For daily backups, use incremental backups to avoid switching tapes during the backup, or to reduce the amount of time spent performing the backup.[2]
Eight hours per day	Daily offline backups.

[1] After-image files let you switch after-image extents without taking the database offline.

[2] The amount of time you save by performing incremental backups depends on the speed of the backup device.

6.5.3 Time Needed For Recovery

How you archive and restore data depends on how long you can spend recovering the database if it becomes unavailable. Table 6–6 provides examples.

Table 6–6: Availability and Recovery Time

If You Can Spend . . .	Then Save Changes In . . .
Four hours recovering	A duplicate database on warm standby. Use roll-forward recovery to keep the standby database current with the production database.
Eight hours recovering	A full backup and one incremental or AI file.
Sixteen hours recovering	A full backup and several incremental backups or AI files.
Twenty-four hours recovering	A full backup and any number of incremental or AI files.

You might decide to perform daily backups of the AI files instead of performing incremental backups. However, backing up the AI files backs up whole transactions, not just the blocks that have changed since the most recent backup. Therefore, performing and restoring AI file backups might require more space and time than incremental backups.

Before deciding to back up the AI files every day, consider that recovering the database from many smaller AI files is more intricate than recovering from few larger AI files.

In addition to the time required to recover the database, you must allow time to repair hardware, file systems, system disks, and other system components. If you can afford no more than eight hours to recover your entire system, you should consider using redundant hardware components, such as mirrored disks, and a high availability system with failover support.

6.5.4 Distributed Transactions Occurring Inconsistently

Two-phase commit ensures that distributed transactions (that is, single transactions involving multiple databases) occur consistently across all databases. Two-phase commit protects against inconsistencies by making sure that all databases commit the transaction, or that none commit. Two-phase commit is not necessary for transactions involving a single database. Table 6–7 provides examples.

Table 6–7: Reliability and Two-phase Commit

If the Applications . . .	Then . . .
Do not perform distributed transactions	Do not implement two-phase commit.
Perform distributed transactions and do not require consistency across databases	Do not implement two-phase commit.
Perform distributed transactions and require consistency across databases	Implement two-phase commit.

6.6 Recovery Guidelines

Follow these guidelines to ensure a safe recovery:

Always:

- Include step-by-step instructions and checklists in the recovery plan.
- Keep a hard copy of the recovery plan.
- Back up the database on a regular basis.
- Back up the AI files on different media from the database and BI files.
- Label, test, and keep the backups.

Do Not:

- Store a BI file on the same disk as an AI file.
- Erase a DB, BI, or AI file unless you are certain you no longer want the database, or unless you have a secure, complete backup.
- Copy a database file without also copying the BI files associated with that database.
- Restore a database file without restoring the corresponding BI file.
- Copy a database with an operating system utility while the database is in use without running PROQUIET.

CAUTION: If you run Progress with the Unreliable Buffered I/O (`-r`) parameter and the system fails because of a system crash or power failure, you cannot recover the database. If you run Progress with the No Crash Protection (`-i`) parameter and the database fails for any reason, you cannot recover the database.

7

Backing Up a Database

Backing up your database is an important part of database maintenance. Regular backups provide a starting point for recovery of a database lost to hardware or software failure. This chapter contains the following sections:

- Performing a Progress Backup
- Performing an Offline Progress Backup
- Performing an Online Progress Backup
- Using Database Quiet Points
- Performing a Non-Progress Backup
- Database Backup Examples
- Verifying a Backup
- CRC Codes and Redundancy In Backup Recovery
- Restoring a Database
- Database Restore Examples

7.1 Performing a Progress Backup

Using the Progress Backup utility (PROBKUP) you can perform an online full backup, an online incremental backup, an offline full backup, or an offline incremental backup. Which you use is determined by your backup plan. See Chapter 6, "Backup and Recovery Strategies," to learn more about creating a backup plan. The syntax below details the parameters to use with PROBKUP:

```
probkup [ online ] db-name [ incremental ] device-name
        [-estimate
        | -vs n
        | -bf n
        | -verbose
```

`online`

Performs an online backup.

db-name

Specifies the database you want to back up.

`incremental`

Performs an incremental backup. If you do not specify this parameter, PROBKUP performs a full backup. Perform a full backup of a database before performing the first incremental backup. Following the first incremental backup, you can perform any number of incremental backups. However, you should perform a full backup at least once a week.

device-name

Identifies a special device (for example, a tape drive) or a standard file. If *device-name* identifies a special device, Progress assumes the device has removable media, such as a tape or a floppy diskette. For Windows NT, use \\.\tape0 for the device name if you are backing up to a tape drive.

`-estimate`

Gives a rough estimate of the amount of media required for offline backups.

NOTE: PROBKUP does not perform a backup when the `-estimate` parameter is used.

`-vs` *n*

Indicates the volume size in database blocks that can be written to each removable volume. Before PROBKUP writes each volume, it displays a message that tells you to prepare the next volume. After writing each volume, a message tells you to remove the volume.

If you use the Volume Size (parameter, the value must be greater than the value of the Blocking Factor (`-bf`) parameter. For example, do not specify a zero value with the Volume Size parameter. See the "UNIX Incremental Backup Example" section for an example of how to use the Volume Size parameter.

If you do not use the Volume Size parameter, PROBKUP assumes there is no limit and writes the backup until completion or until the volume is full. When the volume is full, PROBKUP prompts you for the next volume.

`-bf` *n*

Improves the transfer speed to tape-backup devices by specifying that the data is transferred in amounts optimal for the particular backup device. It indicates the blocking factor for blocking data output to the backup device. The blocking factor specifies how many blocks of data are buffered before being transferred to the backup device. NT uses a variable block size up to 4K. For all other operating systems, each block is the size of one disk block (1K on UNIX). The default for the blocking factor parameter is 34.

`-verbose`

Directs the PROBKUP utility to display information during the backup. If you specify the Verbose parameter, PROBKUP displays "Backed up *n* blocks in *hh*:*mm*:*ss*" every 10 seconds. If you do not specify the Verbose parameter, the message appears only once when the backup is complete.

For more information, see the description of the PROBKUP utility in Chapter 19, "Database Administration Utilities."

7.1.1 Performing an Online Full Backup With PROBKUP

To perform an online, full backup, run PROBKUP from the operating system prompt using the following command syntax:

```
probkup online db-name device-name [ parameters ]
```

UNIX Full Backup Example

The database administrator of Company X's development department performs a full backup of the `devel.db` database every Friday on 9-track tapes.

Company X's DBA prepares the backup media according to the operating system documentation, then follows these steps to perform a full online backup of the `devel.db`:

1 ♦ Verify that the database is not in use by entering the following command:

```
proutil devel -C BUSY
```

2 ♦ Enter the following command to perform a full online database backup:

```
probkup online devel /dev/rrm/0m -vs 35 -bf 20 -verbose
```

These are the parameters for the commands:

`devel`

Identifies the name of the database you are backing up.

`online`

Specifies that the backup is an online backup.

`/dev/rrm/0m`

Specifies the output destination is a tape drive, `/dev/rrm/0m`.

`-vs 35`

Indicates that the volume size in database blocks is 35. If you do not specify the volume size, Progress fills the entire tape before prompting you for a new tape.

`-bf 20`

Specifies that the blocking factor is 20.

`-verbose`

Displays information at 10-second intervals during the installation.

As the full offline backup of `devel.db` begins, the following report appears:

```
64 bi blocks will be dumped.
336 out of 336 blocks in devel will be dumped.
This will require 369664 bytes of backup media.
This backup will require a minimum of 400 blocks to restore.
1 volume will be required.
Backed up 400 blocks in 00:00:05.
Wrote a total of 18 backup blocks using 369664
bytes of media.
Backup complete.
```

The number of backup blocks is the number of `-bf` units written to the tape. Backup blocks contain data, primary recovery, and error-correction blocks.

This example backs up a very small database. Using the `-red` parameter on a larger database increases the amount of time and backup media required for the backup. Also, PROBKUP displays the number of blocks and the amount of backup required for an uncompressed database because you cannot specify the `-scan` parameter for an online backup.

3 ♦ If you enable after-imaging, back up the AI files to a separate tape or disk using a UNIX backup utility.

7.1.2 Progress Backup Options

By default, PROBKUP performs a full backup. To perform an incremental backup, specify the incremental qualifier:

```
probkup online db-name incremental device-name [parameters]
```

`online`

Performs an online backup.

db-name

Specifies the database you want to back up.

`incremental`

Performs an incremental backup. If you do not specify this parameter, PROBKUP performs a full backup. Perform a full backup of a database before performing the first incremental backup. Following the first incremental backup, you can perform any number of incremental backups. However, you should perform a full backup at least once a week.

`device-name`

Identifies a special device (for example, a tape drive) or a standard file. If *`device-name`* identifies a special device, Progress assumes the device has removable media, such as a tape or a floppy diskette. For Windows NT, use \\.\tape0 for the device name if you are backing up to a tape drive.

`parameters`

For a complete list of PROBKUP parameters and more detailed information, see the description of the PROBKUP utility in Chapter 19, "Database Administration Utilities."

7.1.3 Testing Backups

Test backups regularly to ensure they are valid and available when you need them. Be sure to test the backup before you need it to restore a database. Ideally, you should test the backup immediately after you perform it. Then, if there are problems with the backup, you can make another backup copy. If you wait until you need the backup to test it, the database will no longer be available.

Table 7–1 lists the Progress Restore (PROREST) utility parameters you can use to test backups performed with PROBKUP.

Table 7–1: PROREST Utility Verification Parameters

Parameter	Description
-vp	Specifies that the restore utility read the backup volumes, and compute and compare the backup block cyclic redundancy checks (CRCs) with those in the block headers. To recover any data from a bad block, you must have specified a redundancy factor when you performed the database backup. See the "CRC Codes and Redundancy In Backup Recovery" section for more information about error correction blocks and data recovery. You can use the Partial Verify parameter with both online and offline backups.
-vf	Specifies that the PROREST utility will compare the backup block-for-block to the database.

The Partial Verify and Full Verify parameters do not restore or alter the database.

For more information, see the description of the PROREST utility in Chapter 19, "Database Administration Utilities."

If you use an operating system utility to back up the database, with each backup, verify that you have backed up the entire database. The PROBKUP utility automatically backs up the appropriate files; with an operating system utility, you must make sure the files are included in the backup.

7.1.4 Archiving Backups

Properly archiving backups helps you ensure database integrity. Follow these guidelines when archiving backups:

- Clearly label each backup volume. Information on the label should include:
  - The type of backup (incremental or full, online or offline)
  - The date and time of the backup
  - The full pathname of the database
  - The volume number and total number of volumes of the media (volume 1 of 4, for example)
  - The initials of the person who performed the backup
  - The exact command used to back up the database
- Keep a minimum of 10 generations of full backups. Keep daily backups for at least two weeks, weekly backups for at least two months, and monthly backups for a year. Buying extra tapes is much less expensive than manually reconstructing lost databases.
- Keep backups in an area other than where the computer is located, preferably in another building. In the event of building damage, you are less likely to lose both the online and backup versions of the database.

7.2 Performing an Offline Progress Backup

You can perform both full and incremental backups offline. You must perform a full backup before performing an incremental backup. Following the first full backup, you can perform any number of incremental backups. However, you should perform a full backup at least once a week.

Follow these steps to perform an offline backup, either full or incremental:

1 ♦ Verify that the database is not in use.

If you are performing an offline backup, the database must not be in use. The server must be shut down and no single-user session can be active. PROBKUP does not allow access to the database during a full offline backup.

On UNIX systems that support file locking and where backup utilities (for example, `cpio`) honor file locks, an attempt to back up a database in use causes the utility to hang, waiting for the Progress session to end. On all other systems, there is no way to prevent you from backing up a database in use.

Use the PROUTIL BUSY utility to determine whether the database is in use:

```
proutil db-name -C busy
```

The BUSY qualifier returns a code indicating whether the database is in use. You can use the codes returned by the BUSY qualifier in scripts, files, or procedures. For more information about this qualifier, see the description of the PROUTIL utility in Chapter 19, "Database Administration Utilities."

2 ♦ Shut down the database server.

Before you back up the database, you must shut down the database server. See Chapter 5, "Starting Up and Shutting Down," for information about shutting down the database server.

3 ♦ Perform the offline backup.

Use PROBKUP to perform either a full or incremental offline backup. Run PROBKUP from the operating system prompt:

```
probkup db-name device-name [ parameters ]
```

For more information about this qualifier, see the description of PROBKUP parameters listed earlier in this chapter, or see the description of the PROUTIL utility in Chapter 19, "Database Administration Utilities."

By default, PROBKUP performs a full backup. To perform an incremental backup, specify the incremental qualifier.

As you begin the full backup of a database, a report appears on your terminal that indicates how many:

- Bytes are required on the backup media
- Active data blocks are written to the backup media
- Blocks are dumped
- Blocks are required to restore the database

When the backup successfully completes, the report displays the total number of bytes on the backup media and how long it took to complete the backup.

NOTE: If a system failure occurs while you are performing the full backup, perform the backup again.

7.3 Performing an Online Progress Backup

Use PROBKUP to perform either a full or incremental online backup. Run PROBKUP from the operating system prompt. Use the following command syntax:

```
probkup online db-name device-name [ parameters ]
```

By default, PROBKUP performs a full backup. To perform an incremental backup, specify the incremental qualifier:

```
probkup online db-name incremental device-name [ parameters ]
```

7.4 Using Database Quiet Points

If you use OS disk mirroring to provide data redundancy as part of your backup and recovery strategy, you can use database quiet points to maintain database consistency during an OS mirror fracture or split operation on an active online database.

Follow these steps to maintain database consistency during an OS mirror fracture or split operation on an active online database:

1 ♦ Use the PROQUIET command to enable a database quiet point:

```
proquiet dbname enable
```

dbname specifies the name of the database for which you are enabling a database quiet processing point.

NOTE: For more information on and the complete syntax for the PROQUIET command, see Chapter 17, "Startup and Shutdown Commands."

During a database quiet processing point all file write activity to the database is stopped. Any processes that attempt to start a transaction while the quiet point is enabled must wait until you disable the database quiet processing point.

2 ♦ Use an operating system utility to perform the OS mirror fracture or split operation.

Upon successful completion of this command, the fractured disk contains a duplicate of the active online database.

3 ♦ Use the PROQUIET command to disable the database quiet point:

```
proquiet dbname disable
```

dbname specifies the name of the database for which you are disabling the database quiet processing point.

For more information on, and the complete syntax for, PROQUIET, see Chapter 17, "Startup and Shutdown Commands."

4 ♦ Update the structure description (.st) file of the fractured version of the database. Replace the logical location reference (which still references the active database) with the physical location reference of the fractured mirror.

5 ♦ Use the PROSTRCT utility with the REPAIR qualifier to update the shared memory and semaphore identification information to reflect the offline status of the fractured version of the database, and to update the file list information for a database with the information in the updated .st file:

```
prostrct repair dbname [ description-file ]
```

dbname

Specifies the name of the database for which you are repairing the extent list and master block.

description-file

Specifies the name of the structure description (.st) file.

6 ♦ Use the PROBKUP utility with the -norecover startup parameter to back up the fractured version of the database:

```
probkup dbname -norecover
```

dbname

Specifies the name of the fractured version of the database.

NOTE: The -norecover parameter prevents Progress from performing crash recovery or switching to a new AI extent as part of the backup process. Use of the -norecover parameter is noted as an entry in the .lg file.

7.5 Performing a Non-Progress Backup

When performing a backup using an operating system utility instead of the Progress Backup utility, you must perform the following extra steps:

- Be sure to back up all the proper files. For a list of files to back up, see the "Identifying Files For Back Up" section in Chapter 6, "Backup and Recovery Strategies."
- Make sure that the database is not used during the backup. Otherwise, the backup will be invalid. You can do this either by using the PROQUIET command to create a database quiet point or by shutting down the server and making any single-user session inactive.
- After you perform and verify the backup, mark the database as backed up.

Follow these steps:

1 ♦ Shut down the database server.

Before you back up the database, you must shut down the database server. See Chapter 5, "Starting Up and Shutting Down," for information about shutting down the database.

2 ♦ Verify that the database is not in use.

Use the PROUTIL BUSY utility to determine whether the database is in use:

```
proutil dbname -C busy
```

The BUSY qualifier returns a code indicating whether the database is in use. You can use the codes returned by the BUSY qualifier in scripts, files, or procedures. For detailed information, see the description of the PROUTIL BUSY utility in Chapter 19, "Database Administration Utilities."

3 ♦ Make a note of the last entry in the log file. You will use this information later to verify that the database is not used during the backup.

4 ♦ Back up the database.

Use an operating system backup utility to back up the database files. Table 6–2 in Chapter 6, "Backup and Recovery Strategies," lists the backup options available for each operating system. Ensure that your backup technique backs up the entire file. On many UNIX systems, certain utilities (for example, `cpio`) back up only the first part of files that are larger than a specified size (controlled by the ULIMIT parameter). Backups of only the first portion of a database file are of no value.

5 ♦ Verify that the backup is valid.

First, compare the last entry in the log file against the entry you noted in Step 3. If an entry has been added to the log file since you checked in Step 3, the database might have been used. If the database was used during the backup, then the backup is invalid. You must perform another backup.

Second, verify that you have backed up the entire database. The PROBKUP utility automatically backs up the proper files; with an operating system utility, you must make sure the proper files are included in the backup.

6 ♦ Mark the database as backed up.

After you have verified the backup, use the RFUTIL MARK BACKEDUP utility to mark the database as backed up:

```
rfutil dbname -C mark backedup
```

For more information, see the description of the RFUTIL utility in Chapter 19, "Database Administration Utilities."

7.6 Database Backup Examples

This section includes examples of how to use PROBKUP to perform database backups on UNIX and Windows.

7.6.1 UNIX Incremental Backup Example

This example shows how to use all of the possible incremental backup parameters.

Follow these steps to perform an incremental offline backup of the `devel.db` database. To perform an online backup, skip Steps 2 through 4:

1 ♦ Prepare the backup media according to the operating system documentation.

2 ♦ Verify that the database is not in use by entering the following command:

```
proutil devel -C BUSY
```

3 ♦ Shut down the `devel.db` database by using the PROSHUT command:

```
proshut devel
```

4 ♦ Run PROBKUP `-estimate` to determine how much media is necessary for the backup:

```
probkup devel incremental /dev/null -estimate
```

5 ♦ Enter the following command to perform an incremental offline database backup:

```
probkup devel /dev/rrm/0m incremental -vs 4000 -bf 20
    -verbose -io 1 -com -red 5 -scan
```

Enter the following command to perform an incremental online database backup:

```
probkup online devel /dev/rrm/0m incremental -vs 35
    -bf 20 -verbose -io 1 -com -red 5
```

These are the parameters for the commands:

`devel`

Identifies the name of the database you are backing up.

`online`

Specifies that the backup is an online backup.

`/dev/rrm/0m`

Specifies the output destination is a tape drive, `/dev/rrm/0m`.

CAUTION: Do not use the same reel of tape that was used for the full backup.

`incremental`

Specifies that the backup is an incremental backup.

`-vs 35`

Indicates that the volume size in database blocks is 35. If you do not specify the volume size, PROBKUP fills the entire tape before prompting you for a new tape.

`-bf 20`

Specifies that the blocking factor is 20.

`-verbose`

Displays information at 10-second intervals during the installation.

`-io 1`

Specifies that you can lose one incremental backup and still be able to restore the database. Specifies that all blocks that have changed since the backup before the last backup should be archived.

`-com`

Indicates that the data should be compressed before it is written to the tape drive. If you specify the `-com` parameter and do not use `-scan`, PROBKUP displays the number of blocks and the amount of backup media required for an uncompressed database.

-red 5

Specifies that Progress creates one error-correction block for every five blocks that are backed up.

`-scan`

Allows the backup utility to scan the database before backing it up to determine the number of blocks to be backed up.

As the incremental offline backup of `devel.db` executes, the following report appears:

```
64 bi blocks will be dumped.
13 out of 336 blocks in devel will be dumped.
This will require 103424 bytes of backup media.
This backup will require a minimum of 400 blocks to restore.
1 volume will be required.
Backed up 77 blocks in 00:00:01.
Wrote a total of 5 backup blocks using 103424 bytes of media.
Backup complete.
```

The number of backup blocks is the number of -bf units written to the tape. Backup blocks contain data, BI, and error-correction blocks.

This example backs up a very small database. Using the -red parameter on a larger database increases the amount of time and backup media required for the backup.

As the incremental online backup of devel.db executes, the following report appears:

```
Incremental backup started.
Backed up 70 blocks in 00:00:01.
Wrote a total of 3 backup blocks using 103424 bytes of media.
Backup complete.
```

6 ♦ If you have after-imaging enabled, back up the AI files to a separate tape or disk using a backup utility.

7.6.2 Windows Full Backup Example

The database administrator of Company X's development department performs a full backup of the devel.db every Friday, and incremental backups on other weekdays. Both full and incremental backups are done on disk and on a single disk file.

This example shows how to use several of the full backup parameters. Because Windows limits you to nine tokens per command, you cannot use all the parameters with the PROBKUP command. If you want to use all the available parameters, you must run the backup in a procedure file.

When choosing the parameters, consider that the backup takes significantly longer when you use the -com or -red parameters. If you use the -red parameter, the backup also uses more backup media.

Follow these steps to perform an incremental offline backup of the devel.db database. To perform an online backup, skip Steps 2 through 4:

1 ♦ Prepare the backup diskettes according to the operating system documentation.

2 ♦ Verify that the database is not in use by entering the following command:

```
proutil devel -C BUSY
```

3 ♦ Shut down the devel.db database by entering the following command:

```
proshut devel
```

4 ♦ Run PROBKUP -estimate to determine how much media is necessary for the backup, since this is the first time you are making a backup of the database:

```
probkup devel a:\devback -com -red 5 -scan -estimate
```

The following message tells you about the state of your system, and how much media is necessary for backup:

```
devel requires a total of 338 blocks of full backup media.
devel requires a total of 3 blocks of incremental backup media.
devel requires a total of 14 blocks of incremental backup media
     with one overlap.
Total number of allocated blocks = 336
Total number of allocated blocks that are free: 2
Total number of allocated blocks that are used: 334
```

5 ♦ Enter the following command to perform a full offline database backup:

```
probkup devel a:\devback -vs 708 -verbose -com -red 5 -scan
```

Enter the following command to perform a full online database backup:

```
probkup online devel a:\devback -verbose -com -red 5
```

NOTE: You cannot use the -scan parameter for online backups.

These are the parameters for the commands:

devel

Identifies the name of the database you are backing up.

online

Specifies that the backup is an online backup.

`a:\devback`

Specifies the output destination is a file, `\devback`, on the `a:` drive.

`-verbose`

Tells Progress to display information at 10-second intervals during the installation.

`-com`

Indicates that the data should be compressed before it is written to the disk drive. If you specify the `-com` parameter and do not use `-scan`, PROBKUP displays the number of blocks and the amount of backup required for an uncompressed database.

`-red 5`

Creates one error-correction block for every five blocks that are backed up.

`-scan`

Allows the backup utility to scan the database before backing it up to determine the number of blocks to be backed up.

As the full offline backup of `devel.db` executes, the following report appears:

```
64 bi blocks will be dumped.
336 out of 336 blocks in devel will be dumped.
This will require 369664 bytes of backup media.
This backup will require a minimum of 400 blocks to restore.
1 volume will be required.
Backed up 400 blocks in 00:00:04.
Wrote a total of 12 backup blocks using 369664 bytes of media.
Backup complete.
```

As the full online backup of `devel.db` executes, the following report appears:

```
64 bi blocks will be dumped.
336 out of 336 blocks in devel will be dumped.
This will require 369664 bytes of backup media.
This backup will require a minimum of 400 blocks to restore.
Backed up 400 blocks in 00:00:04.
Wrote a total of 12 backup blocks using 369664 bytes of media.
Backup complete.
```

The number of backup blocks is the number of -bf units written to the tape. Backup blocks contain data, primary recovery (BI), and error-correction blocks.

This example backs up a very small database. Using the -red parameter on a larger database increases the amount of time and backup media required for the backup.

6 ♦ If you have after-imaging enabled, back up the AI files onto a separate disk using a separate operating system backup utility.

7.6.3 Incremental Backup Example

This example shows how to use several of the incremental backup parameters. Because Windows limits you to nine tokens per command, you cannot use all the parameters with the PROBKUP command. If you want to use all the available parameters, run the backup in a procedure file.

When choosing the parameters, consider that the backup takes significantly longer when you use the -com or -red parameters. If you use the -red parameter, the backup also uses more backup media.

Follow these steps to perform an incremental offline backup of the devel.db database. To perform an online backup, skip Steps 2 through 4:

1 ♦ Prepare the backup diskettes according to the operating system documentation.

2 ♦ Verify that the database is not in use by entering the following command:

```
proutil devel -C BUSY
```

3 ♦ Shut down the devel.db database by entering the following command:

```
proshut devel
```

4 ♦ Run PROBKUP -estimate to determine how much media is necessary for the backup:

```
probkup devel -estimate
```

5 ♦ Enter the following command to perform an incremental offline database backup:

```
probkup devel d:\devback1 incremental -bf 20 -io 1
```

Enter the following command to perform an incremental online database backup:

```
probkup online devel d:\devback1 incremental -io 1 -com
```

NOTE: You cannot use the -scan parameter for online backups.

These are the arguments and parameters for the commands:

devel

Identifies the name of the database you are backing up.

online

Specifies that the backup is an online backup.

a:\devback

Specifies the output destination is a file, \devback, on the a: drive.

incremental

Specifies that the backup is an incremental backup.

-bf 20

Specifies that the blocking factor is 20 to match the blocking factor of the disk drive.

-io 1

Specifies that you can lose one incremental backup and still be able to restore the database.

-com

Indicates that the data should be compressed before it is written to the disk drive. If you specify the -com parameter and do not use -scan, PROBKUP displays the number of blocks and the amount of backup required for an uncompressed database.

As the incremental offline backup of `devel.db` executes, the following report appears:

```
The bi file requires a total of 64 blocks of backup media.
sales requires an unknown amount of blocks for backup media.
An unknown number of volumes are required.
Backed up 77 blocks in 00:00:01.
Wrote a total of 4 backup blocks using 82944 bytes of media.
Backup complete.
```

Because you did not specify the `-scan` parameter, Progress does not scan the database before beginning the backup, and therefore cannot determine how many blocks will be backed up. The number of backup blocks is the number of `-bf` units written to the tape. Backup blocks contain data, primary recovery (BI), and error-correction blocks.

As the incremental online backup of `devel.db` executes, the following report appears:

```
Incremental backup started.
Backed up 70 blocks in 00:00:01.
Wrote a total of 3 backup blocks using 103424 bytes of media.
Backup complete.
```

6 ♦ If you have after-imaging enabled, back up the AI files to a separate disk using a separate operating system backup utility.

7.7 Verifying a Backup

Immediately after backing up the database, verify that the backup does not contain any corrupted blocks. Use the Progress Restore utility to verify the integrity of a full or incremental backup of a database as follows:

- Run the Progress Restore utility with the Partial Verify (-vp) parameter. With this parameter, Progress checks the backup for bad blocks and reports whether any exist.
- Run the Progress Restore utility with the Full Verify (-vf) parameter. With this parameter, Progress compares the backup to the database block-for-block.

These parameters do not actually restore the database. They only verify the status of the backup, notify you if there are any bad blocks, and report whether the blocks are recoverable. You must run the restore utility again (without the partial or full verify parameters) to restore the database.

When you use the -vp parameter, PROREST scans the backup and recalculates the CRC code for each block. It then compares the newly calculated CRC code with the CRC code stored in the block header. If the codes do not match, Progress marks the block as bad and displays the following message:

```
CRC check failed reading backup block n
```

If the backup contains error-correction blocks and a redundancy set contains only one bad block, PROBKUP uses the error-correction block (and the other blocks in the redundancy set) to re-create the bad block. The error-correction block is the EXCLUSIVE OR of the backup blocks in the redundancy set. When Progress recovers the block, the following message appears:

```
Recovered backup block n
```

If the redundancy set contains more than one bad block or if the backup does not include error-correction blocks, Progress cannot recover the bad block and displays the following message:

```
n CRC error within recovery group - recovery impossible
```

Progress also cannot recover a corrupted block if the error-correction block itself has a CRC check failure. In this case, the following message appears:

```
Unable to recover previous block in error
```

If Progress encounters 10 unrecoverable errors during the verify pass or during the database restore, you can terminate the verify operation:

```
10 read errors have occurred.
Do you want to continue? [y/n]
```

7.8 CRC Codes and Redundancy In Backup Recovery

To recover corrupted backup blocks, Progress relies on:

- CRC codes to identify bad blocks. A CRC code is automatically calculated for each Progress backup block whether or not you specify a redundancy factor.
- Error-correction blocks to recover bad blocks. Error-correction blocks are included in the backup only if you explicitly request them with the `-red` parameter of the backup utility.

7.8.1 CRC Codes

When Progress writes a block of data to the backup media, it calculates a CRC code based on the contents of the block and stores it with the block. When restoring, Progress reexamines the contents of the block and verifies that they are consistent with the accompanying CRC code. If the block contents are not consistent with the CRC code, the backup block is corrupted.

If the backup includes error-correction blocks, Progress automatically uses the information in those blocks to recover the corrupted block. If the backup does not include error-correction blocks, Progress cannot recover the corrupted block when you restore the database.

7.8.2 Error-correction Blocks

Error-correction blocks contain information about the preceeding set of backup blocks and allow Progress to recover corrupted blocks in a backup. The error-correction block and the blocks it is based on are called a *redundancy set*. You can provide error-correction blocks in the backup by specifying the `-red` parameter in the backup command.

The `-red` parameter specifies a redundancy factor. The redundancy factor determines how many backup blocks are in each redundancy set. For example, if you specify a redundancy factor of 2, Progress creates an error-correction block for every two backup blocks. Therefore, every redundancy set contains two backup blocks and an error-correction block.

Progress can recover a bad backup block if it is the only corrupted block in the redundancy set. If a redundancy set contains more than one bad backup block or a bad backup block and a bad error-correction block, Progress cannot recover any of the bad blocks in the redundancy set.

If you specify a very low redundancy factor (for example, 2), the chance of having two or more bad database blocks in a redundancy set is low. If you specify a higher redundancy factor, the chances are higher. However, lower redundancy values also produce larger backups that require more time and media. If the backup media is highly reliable, you might use a high redundancy factor; if the media is less reliable, you might want to specify a lower redundancy factor.

The size of each backup block—and therefore of each error-correction block—is determined by the `-bf` parameter. The default blocking factor is 34. For example, if the database block is 1,024 bytes and the blocking factor is 40, each backup block is 40K; that is, the size of 40 database blocks.

7.9 Restoring a Database

In the event of database loss or corruption, you can restore the database from a backup. You must restore a database with the same version of Progress that you used to create the backup. For example, you cannot restore backups created with Version 6 with the Version 7 Restore utility. Nor can you back up a Version 6 database with the Version 7 backup utility.

This section describes:

- How to use the PROREST utility to restore a database
- Important rules for restoring backups
- Operating-system-specific examples of database restores

7.9.1 Using the PROREST Utility To Restore a Database

Use the PROREST utility to restore a full or incremental backup of a database:

```
prorest dbname device-name {-list | -vp | -vf}
```

dbname

Specifies the name of the database where you want to restore the backups.

device-name

Identifies the directory pathname of the input device or standard file from which you are restoring the data.

-list

Provides a description of all application data storage areas contained within a database backup. Use the information to create a new structure description file and database so you can restore the backup. For additional information, see the "Obtaining Storage Area Descriptions Using PROREST" section later in this chapter.

-vp

Specifies that the restore utility reads the backup volumes and computes and compares the backup block cyclical redundancy checks (CRCs) with those in the block headers.

To recover any data from a bad block, you must have specified a redundancy factor when you performed the database backup. See the "Error-correction Blocks" section for more information about error-correction blocks and data recovery.

-vf

Specifies that the restore utility compares the backup to the database block-for-block. Do not compare the backup to a database that is in use.

NOTE: When you specify the -vp or -vf parameter, PROREST does not actually restore the database. You must restore the database in a separate step.

The first time you start the database after restoring an online backup, Progress runs normal crash recovery to make sure that any uncompleted transactions at the time of the backup are discarded.

When you restore a full database backup, consider restoring the backup to a new database. This allows you access to the corrupted database, if necessary. You must restore an incremental database backup to a restored database.

If PROREST encounters corrupted backup blocks that it is unable to recover, you lose the data in the corrupted blocks. The amount of lost data is approximately equal to the number of bad blocks multiplied by the blocking factor.

As you begin the restore procedure for a database, a report appears that indicates the date of the backup and the number of blocks required to restore the database.

7.9.2 Important Rules For Restoring Backups

There are several important rules you must follow when you are restoring an incremental backup:

- If you restore over an existing database, verify the tapes before doing the restore. If the existing database is the only copy, back up the existing database before doing the restore.
- Restore a backup with the same Progress version that you used to perform the backup.
- You must restore an incremental database backup to an existing database.
- Create the void database before you restore the backup, or else use the existing structure, overwriting it.
- You must restore a database in the same order that you backed it up. You must first restore the full backup, followed by the first incremental backup, followed by the second incremental backup, etc. If you try to restore a database out of sequence, you get an error message and the restore operation fails.
- If you lose the second incremental and you used an overlap factor of 1, the third incremental correctly restores the data lost in the second incremental.
- After you restore a full backup, do not use the database if you want to restore successive incremental backups. If you make any database changes before completely restoring all backups, any successive, incremental backups (that were not restored) are rejected unless you restart the restore procedure beginning with the full backup.
- If a system failure occurs while you are restoring the database, restart the restore operation beginning with the backup volume that you were restoring at the time of the system failure.
- If a target database exists, it must have the same block size and storage area configuration as the source database. The PROREST utility attempts to expand storage areas to allow a complete restore, but if the storage areas cannot expand, the restore fails.

7.9.3 Obtaining Storage Area Descriptions Using PROREST

Use the PROREST utility with the `-list` parameter to obtain a description of the application data storage areas within a database backup:

```
prorest db-name device-name -list
```

The following example shows the output from the `prorest -list` command:

```
PROGRESS version 9. as of Thu Jan 19 2000

Area Name: Schema Area
       Size: 7168, Records/Block: 32, Area Number: 6
Area Name: Employee
       Size: 2784, Records/Block: 32, Area Number: 7
Area Name: Inventory
       Size: 9952, Records/Block: 32, Area Number: 8
Area Name: Cust_Data
       Size: 2784, Records/Block: 32, Area Number: 9
Area Name: Cust_Index
       Size: 2784, Records/Block: 32, Area Number: 10
Area Name: Order
       Size: 20960, Records/Block: 32, Area Number: 11
Area Name: Misc
       Size: 2784, Records/Block: 32, Area Number: 12
```

Use this information to create a new structure description file and database so you can restore the backup.

7.9.4 Database Restore Examples

This section includes examples of database restores using PROREST for UNIX and Windows.

UNIX Full Backup Restore Example

The database administrator of Company X's Development department wants to restore the devel.db database that was previously backed up.

Follow these steps to restore the devel.db database to a new database from a full backup:

1 ♦ Enter the following command:

```
prorest newdev /dev/rrm/0m
```

The newdev.db database is an empty database. The 9-track tape drive (/dev/rrm/0m) specifies the device from which the full backup is being restored. As the restore begins, the following report appears:

```
This is a full backup of /usr1/develop/devel.db. (6759)
This backup was taken Wed Nov 18 15:34:43 1999. (6760)
The blocksize is 1024. (6990)
It will require a minimum of 3065 blocks to restore. (6763)
Read 41 blocks in 00:00:02
```

This command restores the database devel.db from a tape to newdev.db. The report indicates that volume 1 is being processed.

2 ♦ Use newdev.db with Progress once the restore is complete.

UNIX Incremental Backup Restore Example

If you want to restore an incremental backup of the devel.db database to a new database, you must first restore a full backup, then follow these steps:

1 ♦ Enter the following command to run an incremental restore of the database from a tape once the full restore is done:

```
prorest newdev /dev/rrm/0m
```

The following report appears as the restore begins:

```
This is an incremental backup of /usr1/develop/devel.db. (6759)
This backup was taken Wed Nov 18 15:41:47 1999. (6760)
The blocksize is 1024. (6990)
It is based on the full backup of Wed Nov 18 15:34:43 1999. (6761)
It will require a minimum of 3065 blocks to restore. (6763)
Read 41 blocks in 00:00:00
```

2 ♦ Once the restore is complete, use `newdev.db` with Progress.

Windows Full Backup Restore Example

The database administrator of Company X's development department wants to restore the `devel.db` database.

Follow these steps to restore the `devel.db` database to a new database from a full backup:

1 ♦ Enter the following command:

```
prorest newdev d:\devback
```

The `newdev.db` database is an empty database. The name of the device is `d:` and `\devback` specifies the file from which the full backup is being restored. As the restore begins, the following report appears:

```
This is a full backup of d:\deve1.db. (6759)
This backup was taken Wed Nov 18 15:56:28 1999. (6760)
The blocksize is 4096. (6994)
It will require a minimum of 1337 blocks to restore. (6763)
Read 1274 blocks in 00:00:02
```

This command restores the database `devel.db` from a diskette to `newdev.db`. The report indicates that volume 1 is being processed.

2 ♦ Once the restore is complete, use `newdev.db` with Progress.

Windows Incremental Backup Restore Example

If you want to restore an incremental backup of the `devel.db` database to a new database, you must first restore a full backup, then follow these steps:

1 ♦ Enter the following command to run an incremental restore of the database from a tape once the full restore is complete:

```
prorest newdev e:\devback1
```

The `newdev.db` database is an empty database. The name of the device is `a:` and `\devback` specifies the file from which the full backup is being restored. As the restore begins, the following report appears:

```
This is an incremental backup of D:\deve1.db. (6759)
This backup was taken Wed Nov 18 15:56:28 1999. (6760)
The blocksize is 4096. (6994)
It is based on the full backup of Wed Nov 18 15:56:28 1999 (6761)
It will require a minimum of 1337 blocks to restore. (6763)
Read 1274 blocks in 00:00:02
```

2 ♦ Once the restore is complete, use `newdev.db` with Progress.

8

Recovering a Database

This chapter describes the different ways to recover your Progress database and transactions if your system or disks fail.

Specifically, this chapter contains the following sections:

- Introduction To Recovery Mechanisms
- File Locations That Ensure Safe Recovery
- Sample Recovery Plans
- After-imaging and Roll-forward Recovery Commands
- Recovering From System Failures
- Recovering From Media Failures
- Recovering From a Full Disk
- Truncating the BI File
- Recovering From a Crash
- Recovering From a Lost Or Damaged Control Area
- Unlocking Damaged Databases
- Dumping Tables From a Damaged Database
- Forcing Access To a Damaged Database With the -F Parameter

8.1 Introduction To Recovery Mechanisms

The Progress Version 9 database has three kinds of recovery mechanisms:

- **Crash recovery** — Uses primary recovery (BI) data to recover from system failures
- **Roll-forward recovery** — Uses backups and after-image data to recover from media failures
- **Two-phase commit** — Ensures that transactions occur consistently across multiple databases

Depending on your site requirements, you might choose not to implement all three of these recovery mechanisms. Figure 8–1 shows the order of precedence of these mechanisms. Crash recovery requires use of a recovery (BI) log and occurs without any interaction. Roll-forward recovery requires use of an after-image (AI) log. Two-phase commit requires use of a transaction log (TL). If you use two-phase commit, be sure to also use after-imaging.

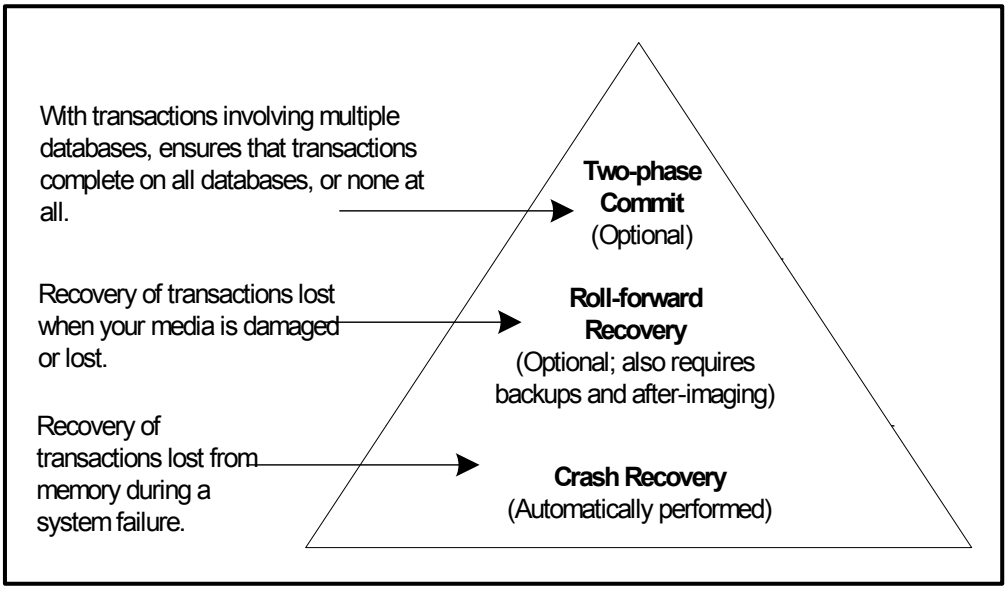


Figure 8–1: Progress Recovery Mechanisms

Each mechanism relies on notes that are written to a file to record database changes. A *note* is a record of the smallest unit of change in a database. For example, a record of one change made to one block in the database. The database engine automatically records database changes as notes in the primary recovery (BI) log. If after-imaging is enabled, it also records notes to the after-image (AI) log. If two-phase commit is enabled, it also records transactions and notes to the transaction log (TL).

8.1.1 Crash Recovery

Crash recovery occurs automatically. With this feature, the database engine uses information from the primary recovery (BI) log to recover from system failures.

The BI files are a vital part of the database. You should treat the files as an integral part of the database unit. When you back up and restore the database, back up and restore the DB and BI files together. Never manually delete the BI files.

While the database is running, database transaction information exists in three areas:

- In the database on disk
- In the buffer pool in memory
- In the BI files on disk

When database records are modified, the changes occur first in memory. When a transaction is committed, the change is recorded to the BI file. Over time, the database engine makes the change to the database file on disk. If the system fails, the information stored in the buffer pool is lost. The database engine performs crash recovery using the information logged to the BI file to re-create lost transactions and undo transactions that were not committed.

Before updating the database, the database engine makes a copy of the current information and writes it to the BI file. This activity begins at the end of an update operation. If the system fails during the transaction, the engine uses the information in the BI file to restore the database to its pretransaction state. The engine also uses the information in the BI files during normal processing to undo transactions.

For example, suppose you execute the following Progress 4GL procedure:

```
FOR EACH customer:
  UPDATE name max-credit.
END.
```

You update customers 1 and 2, and while you are updating customer 3, the system fails. When you restart the database, messages appear in the database .lg file similar to the following:

```
11:13:54 Single-user session begin for marshall on /dev/pts/25 (451)
11:13:54 Begin Physical Redo Phase at 256 . (5326)
11:13:56 Physical Redo Phase Completed at blk 800 of 8165 and 31829 (7161)
11:13:56 Begin Physical Undo 1 transactions at blk 800 offset 8189 (7163)
11:14:38 Physical Undo Phase Completed at 1020 . (5331)
11:14:38 Begin Logical Undo Phase, 1 incomplete transactions are being backed
out. (7162)
11:14:38 Logical Undo Phase Complete. (5329)
11:14:46 Single-user session end. (334)
```

The messages indicate the necessary phases of crash recovery performed by the database engine to bring the database to the consistent state that existed prior to the system failure. Since the engine performs crash recovery every time you open the database, not all of the recovery phases are logged in the database .lg file. For example, the engine performs and logs the Physical Redo phase unconditionally, but the Physical Undo and Logical Undo phases are only performed and logged when outstanding transactions are found.

When you rerun the same procedure, customers 1 and 2 remain updated, but the database engine has used the BI file to restore customer 3 to the state it was in before you began the update.

Crash recovery protects you from system failures, but it does not protect you from loss of media. In the event of media loss, you must restore from a backup, and either manually re-enter the lost transactions or use the roll-forward recovery mechanism to re-create the transaction.

8.1.2 Roll-forward Recovery

Roll-forward recovery, used together with after-imaging and backup, lets you recover from media failures. When a database disk fails, you can restore the most recent backup, then use roll-forward recovery to restore the database to the condition it was in before you lost the disk. With roll-forward recovery, the database engine uses data in the AI files to automatically reprocess all the transactions that have been executed since the last backup was made.

To use roll-forward recovery, you must:

- Perform regularly scheduled backups of the database. Regular backups are a fundamental part of recovery. For detailed information on backups and developing a backup schedule, see Chapter 6, “Backup and Recovery Strategies.”
- Enable after-imaging immediately after you complete the backup. See Chapter 11, “After-Imaging,” for information about enabling after-imaging.

- Perform regularly scheduled backups of the AI files. See Chapter 7, "Backing Up a Database," for information about backing up the AI files.
- Store the AI files on different disks than those containing the database and BI files.

  When you enable after-imaging, the database engine writes database changes to the AI files. If you store the AI files on the same disks as the database or BI files and a disk is corrupted, you cannot use the AI files to recover that database.
- Archive the AI files to tape or other durable media as they become full.

This example shows how the database engine uses the AI files to restore the database. Suppose you execute the following Progress 4GL procedure:

```
FOR EACH customer:
  UPDATE name max-credit.
END.
```

You update customers 1 and 2, and while you are updating customer 3, the disk where the database file is stored is damaged. You cannot use the BI file to restore the transactions because the original database is no longer valid.

However, because you enabled after-imaging, you can use roll-forward recovery to recover the database. If you do not enable after-imaging, you lose all the updates since the last database backup.

Before updating the database, the database engine makes a copy of the current information and writes it to the BI file and the AI file.

After updating customers 1 and 2, the database disk fails while updating customer 3. The AI files have a copy of all transactions completed since the last backup. Restore the last backup of the database and then roll forward the AI files to produce a restored database that contains all completed transactions.

8.1.3 Two-phase Commit

Two-phase commit ensures that distributed transactions (that is, transactions involving multiple databases) occur consistently across all databases. Two-phase commit is not necessary for transactions involving a single database.

For detailed information on two-phase commit, see Chapter 12, "Using Two-phase Commit."

NOTE: If you use two-phase commit, you should also use after-imaging to ensure database integrity and avoid backup synchronization problems. Although two-phase commit ensures that two databases remain synchronized, it does not protect you from a lost database or BI file. If you lose an entire file or disk, you must use the AI file to roll forward to the point of the crash.

8.2 File Locations That Ensure Safe Recovery

The AI files, together with the most recent database backup, contain the same information as the DB files. If you lose the disk containing the DB and BI files, you can use the AI files and database backup to reconstruct the database. For this reason, you should place the AI files on a different disk from the one that contains the DB and BI files.

Figure 8–2 shows sample locations for each of the DB, BI, AI, and event log files.

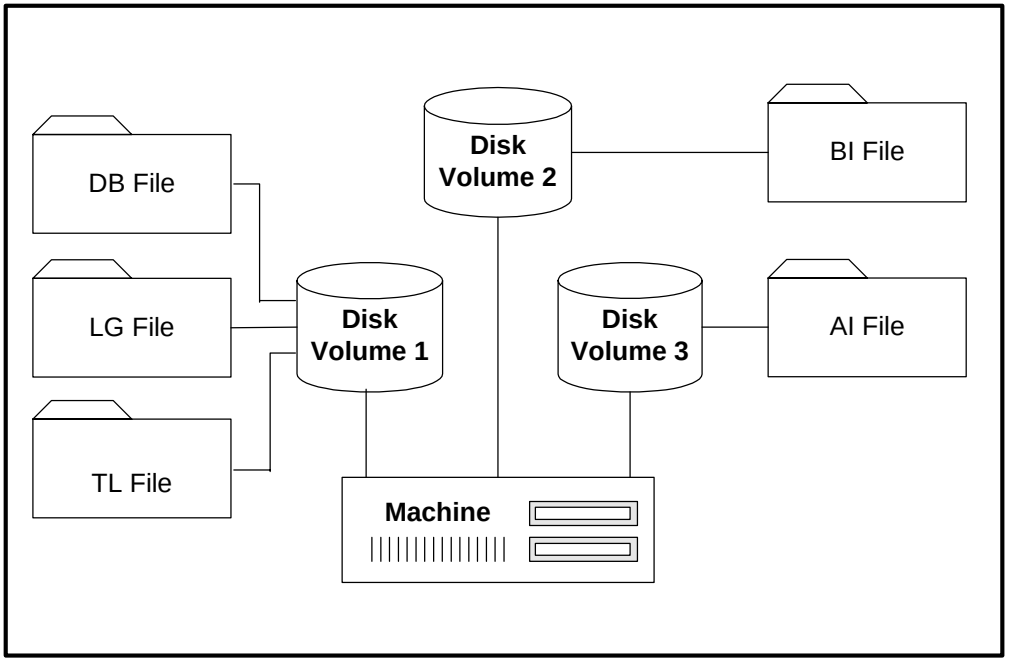


Figure 8–2: Storing Database Files

8.3 Sample Recovery Plans

This section provides sample recovery plans to meet a range of availability requirements, including low, moderate, moderately high, and high availability requirements.

8.3.1 Example 1: Low Availability Requirements

The database administrators at site A have determined the following availability requirements. Table 8–1 provides examples.

Table 8–1: Sample Low Availability Requirements

Question	Answer	Recovery Technique
How many committed transactions can you afford to lose?	1 week of transactions	Perform weekly backups.
How long can the application be offline while you perform scheduled maintenance?	8 hours per day	Perform weekly backups offline.
How long can you spend recovering if the database becomes unavailable?	24 hours	Keep a full backup and any number of incrementals or AI files.
Must distributed transactions occur consistently across databases?	No	Do not implement two-phase commit.

Given these requirements, the database administrators perform a full offline backup every Monday at 5 PM. Incrementals are not required because the site can afford to lose one week of transactions, and full backups are performed weekly. The backup is tested Tuesday at 5 PM using disk space reserved for that purpose.

8.3.2 Example 2: Moderate Availability Requirements

The database administrators at site B have determined the following availability requirements. Table 8–2 provides examples.

Table 8–2: Sample Moderate Availability Requirements

Question	Answer	Recovery Technique
How many committed transactions can you afford to lose?	1 day of transactions	Perform daily backups.
How long can the application be offline while you perform scheduled maintenance?	8 hours per week	Perform weekly offline backups with daily online full or incremental backups.
How long can you spend recovering if the database becomes unavailable?	8 hours	Keep a full backup and one incremental or AI file.
Must distributed transactions occur consistently across databases?	No	Do not implement two-phase commit.

Given these requirements, the database administrators perform a full offline backup every Saturday night. Because they can lose only one day of transactions, they supplement weekly backups with daily online incremental backups. Because they can only allow time to restore a single incremental, they perform incremental backups with an *overlap factor* of six. This means that each incremental backup saves changes that were saved in the last six incremental backups. The full backup is tested Saturday night, and after the incremental backups finish they are tested using disk space reserved for that purpose.

8.3.3 Example 3: Moderate-to-High Availability Requirements

The database administrators at site C have determined the following availability requirements. Table 8–3 provides examples.

Table 8–3: Sample Moderate-to-High Availability Requirements

Question	Answer	Recovery Technique
How many committed transactions can you afford to lose?	A few minutes of transactions	Enable after-imaging.
How long can the application be offline while you perform scheduled maintenance?	Never	Perform online backups with after-image files.
How long can you spend recovering if the database becomes unavailable?	24 hours	Restore a full backup and a series of any number of incrementals or AI files.
Must distributed transactions occur consistently across databases?	Yes	Implement two-phase commit.

Given these requirements, the database administrators perform a full online backup every Saturday night. Because they can afford to lose only a few minutes of transactions, the site also enables after-imaging. They switch AI extents every day at 7 PM and archive the resulting full after-image extent. (Because they switch AI extents at scheduled intervals rather than waiting for an extent to fill, they must be sure to monitor AI extent usage in case an extent fills before the scheduled switch.) Because applications at this site perform distributed transactions that must be consistent across databases, this site implements two-phase commit. The full backup is tested Saturday night. After they are archived, AI files are applied to the tested backup since so few transactions can be lost.

8.3.4 Example 4: High Availability Requirements

The database administrators at site D have determined the following availability requirements. Table 8–4 provides examples.

Table 8–4: Sample High Availability Requirements

Availability Question	Answer	Recovery Technique
How many committed transactions can you afford to lose?	None	Enable after-imaging.
How long can the application be offline while you perform scheduled maintenance?	Never	Perform online backups with after-image files.
How long can you spend recovering if the database becomes unavailable?	4 hours	Keep a duplicate system on warm standby. Use roll-forward recovery to keep the standby database current with the production database.
Must distributed transactions occur consistently across databases?	No	Do not implement two-phase commit.

Given these high availability requirements, the database administrators keep a duplicate database on warm standby. They follow these steps:

1 ♦ Restore a backup of the production database to an empty database with a duplicate structure on a different system.

2 ♦ On the production database, enable after-imaging using AI files with fixed-length extents.

3 ♦ On the production database, whenever a fixed-length AI extent becomes full, copy it to the standby system and roll it forward on the standby database.

4 ♦ After bringing the standby database up to date, mark the full after-image extent on the production database as empty to make it available for reuse.

In addition, backups and AI files are constantly tested and verified on the standby database. The standby database is put through crash recovery, verified, and if possible, backed up before restoring online backups of the production database to the standby database.

8.3.5 Sample Recovery Scenarios

To show how the parts of a recovery plan fit together, this section presents an overview of the steps and commands you use to implement a database recovery plan.

Figure 8–3 through Figure 8–7 show sample scenarios of a one-week period.

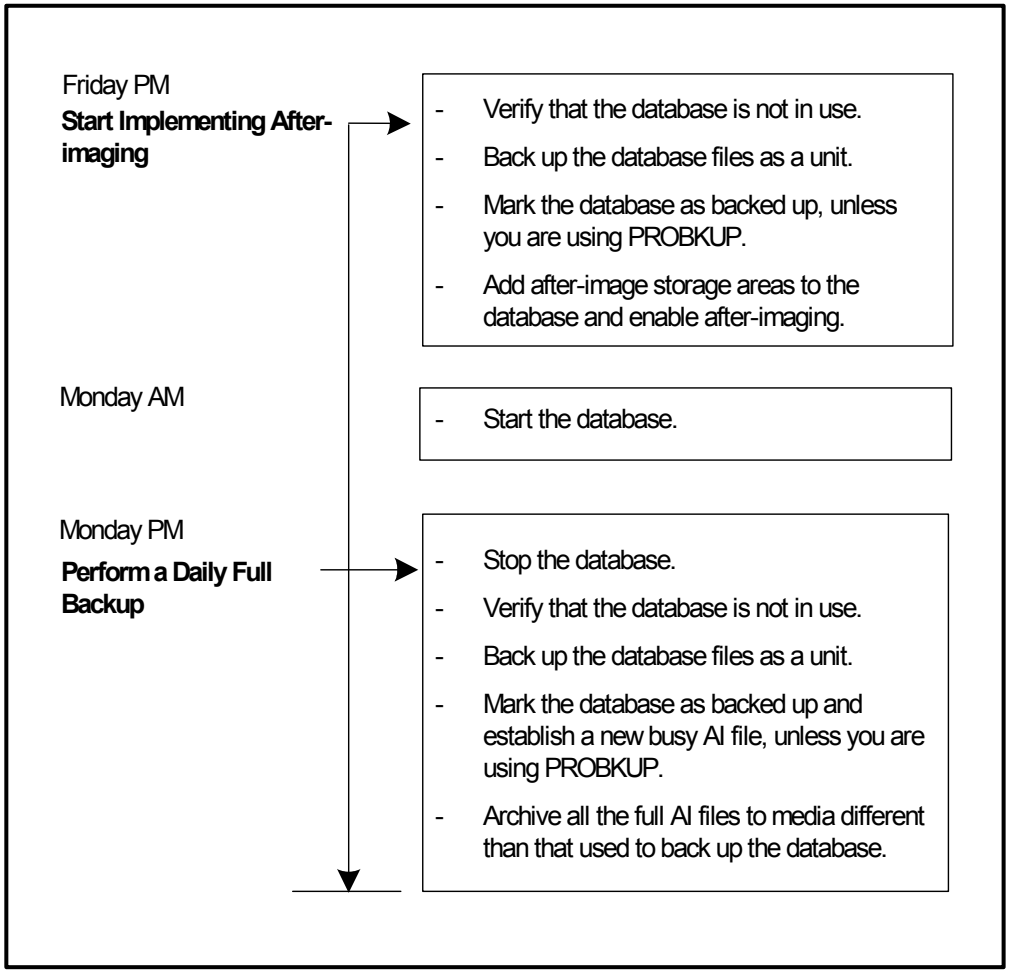


Figure 8–3: Recovery Scenario - Day 1

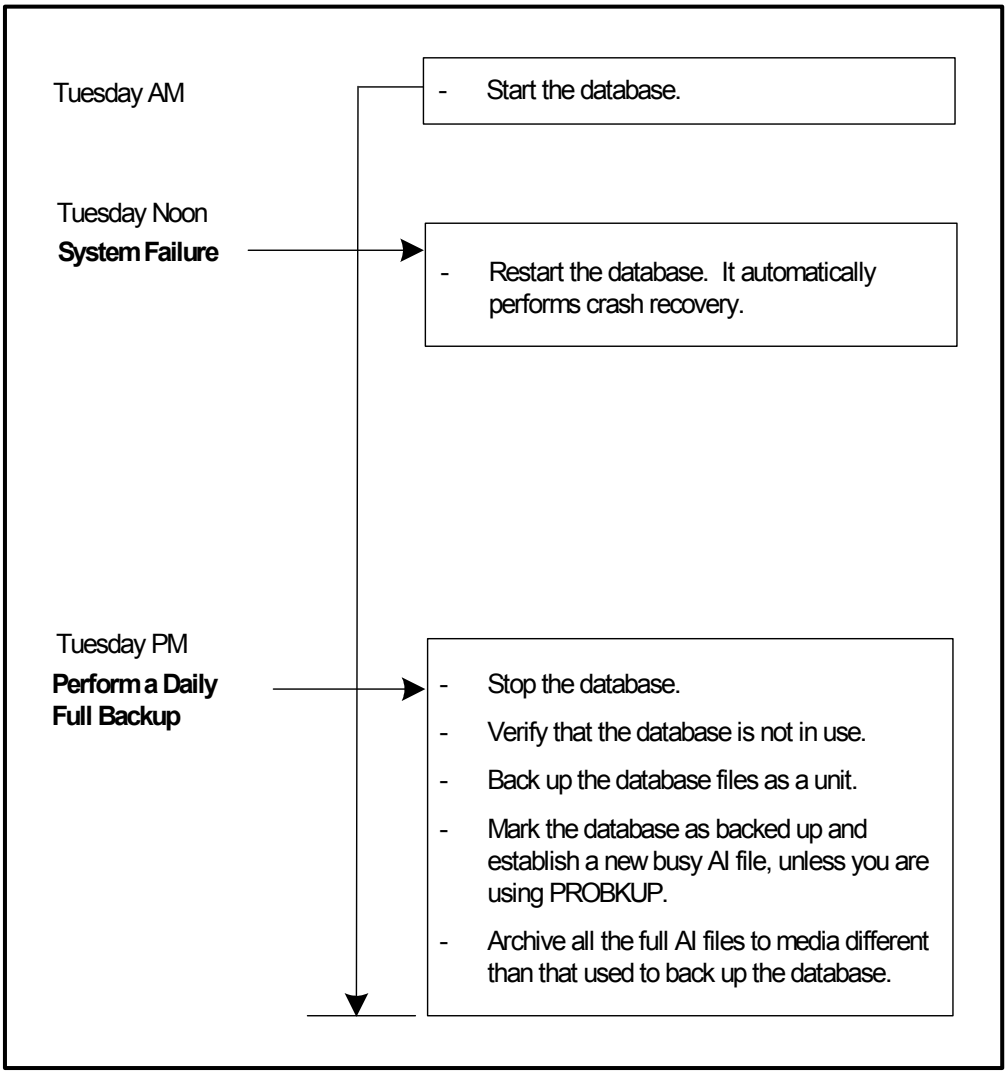


Figure 8–4: Recovery Scenario - Day 2

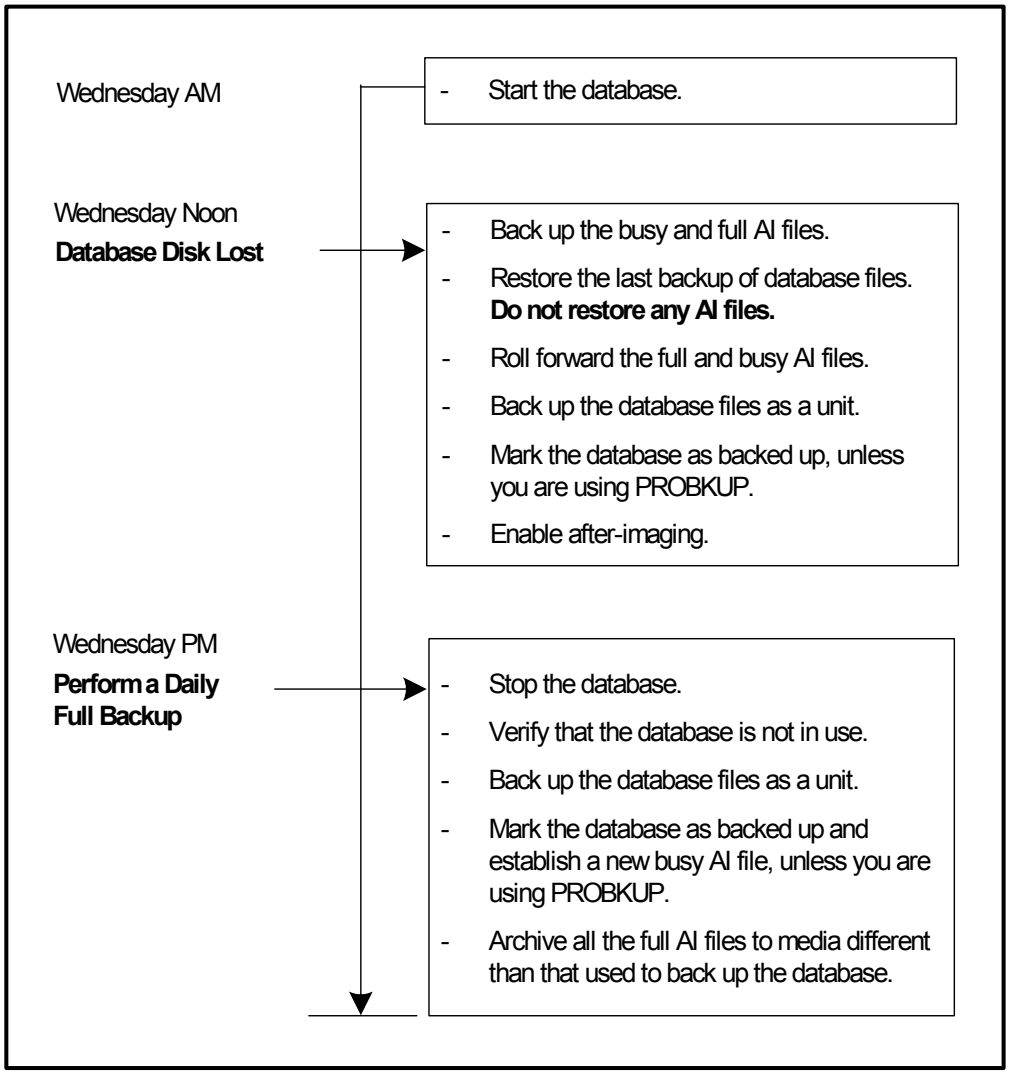


Figure 8–5: Recovery Scenario - Day 3

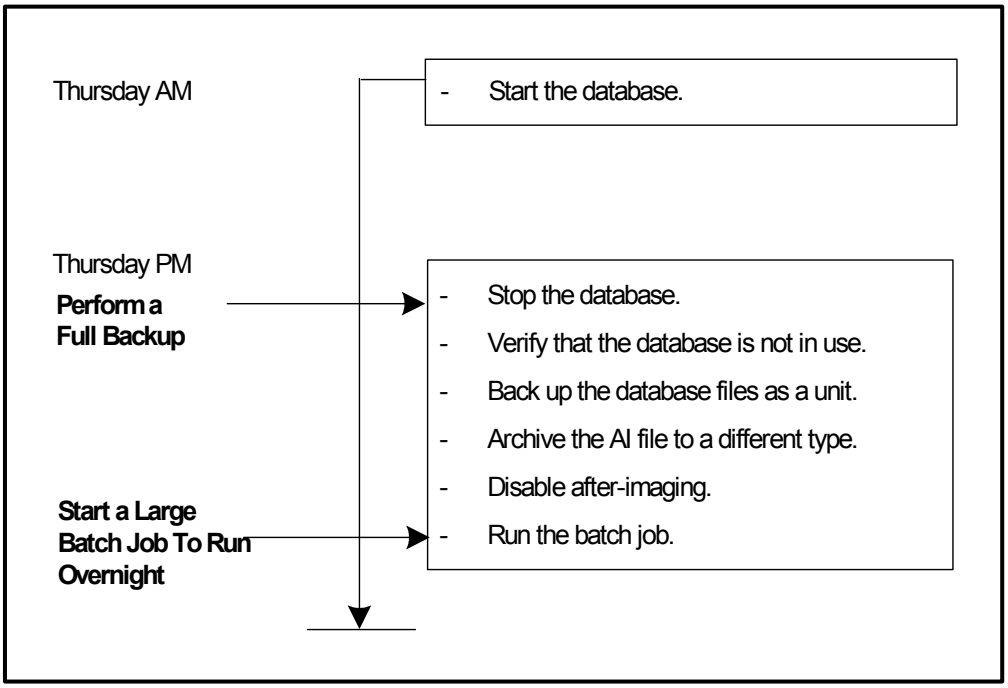


Figure 8–6: Recovery Scenario - Day 4

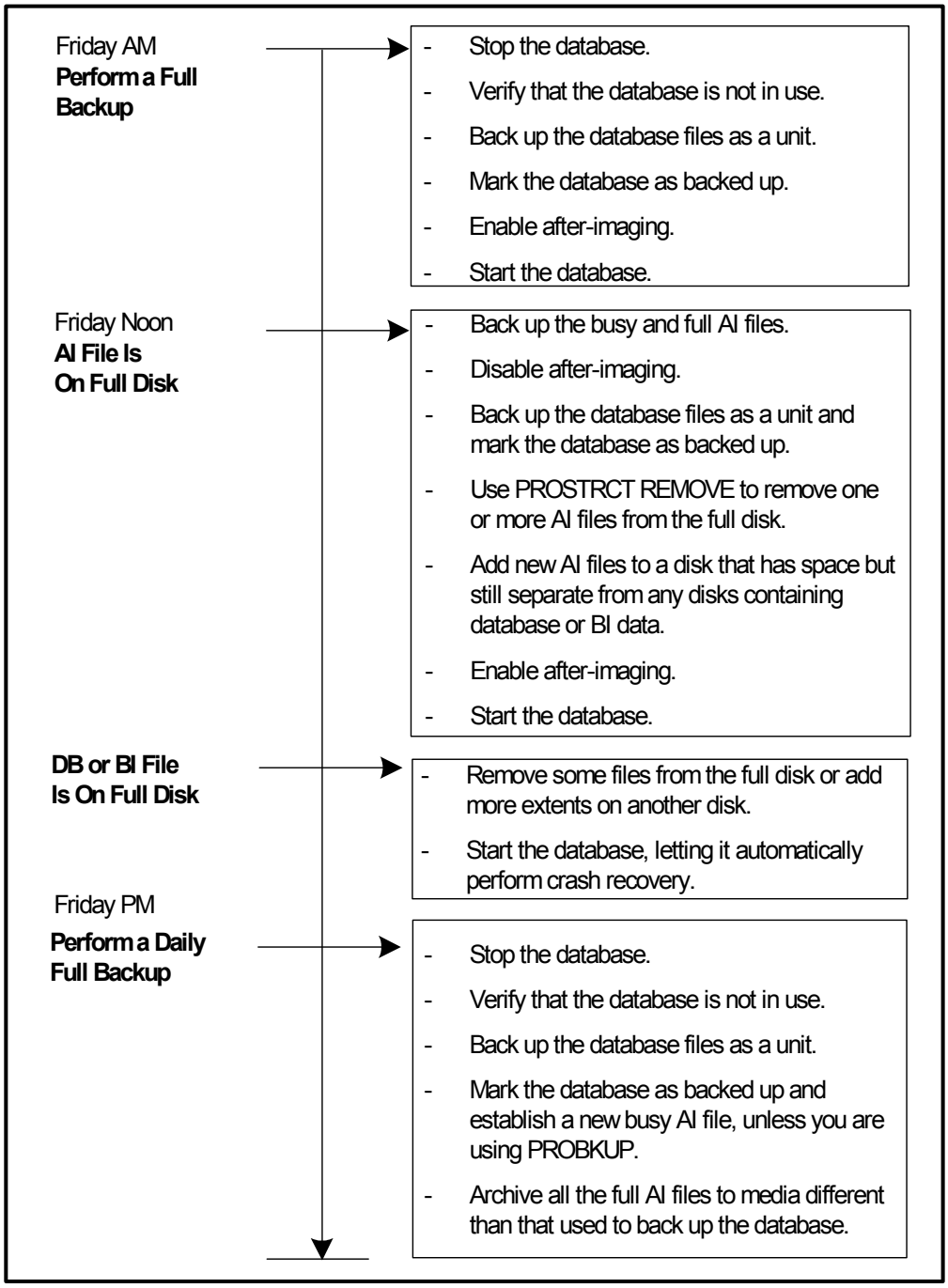


Figure 8–7: Recovery Scenario - Day 5

8.4 After-imaging and Roll-forward Recovery Commands

Table 8–5 lists the utilities you use to perform the steps in the scenarios shown in Figure 8–3 through Figure 8–7. These utilities are described in detail in Chapter 19, "Database Administration Utilities."

Table 8–5: Utilities Used With Roll-forward Recovery

Function	Utility Command
Verify that the database is not in use. The database cannot be in use when you perform a full offline backup.	PROUTIL BUSY
Back up the database. You must perform a full backup of the database before you can use roll-forward recovery.	PROBKUP or operating system utility
Mark the database as backed up. If you do not use PROBKUP to back up the database, you must mark the database as backed up to begin after-imaging or start a new after-image file.	RFUTIL MARK BACKEDUP
Start a new AI file. After you back up the database and AI files, you can truncate the existing AI file.	RFUTIL AIMAGE NEW
Disable after-imaging. Before you run a batch job, you might want to disable after-imaging to improve performance.	RFUTIL AIMAGE END
Restart the roll-forward operation. Use of this qualifier is limited to RFUTIL roll-forward operations that fail because of power outages or system failures.	RFUTIL ROLL FORWARD RETRY

8.5 Recovering From System Failures

Before-imaging is automatically enabled to let you recover from a system failure. If the system fails, the technique for recovering the database depends on the database operation you were performing when the system crashed. The following sections provide specific information.

8.5.1 System Crash While Running RFUTIL ROLL FORWARD

If the system crashes while you are using the RFUTIL Roll Forward utility, use the RETRY qualifier to resume the roll forward:

```
rfutil db-name -C roll forward retry
```

Use of this qualifier is limited to RFUTIL roll-forward operations that fail because of power outages or system failures. The ROLL FORWARD RETRY qualifier restarts the roll-forward operation on the after-image extent that was in the process of rolling forward. The retry operation finds the transaction in process at the time of failure and resumes rolling forward.

8.5.2 System Crash While Running Other Utilities

If the system crashes while you are using any of the following utilities, simply rerun the utility:

- PROUTIL BUSY
- PROUTIL HOLDER
- PROUTIL TRUNCATE BI
- RFUTIL AIMAGE BEGIN
- RFUTIL AIMAGE END
- RFUTIL AIMAGE NEW
- RFUTIL MARK BACKEDUP
- RFUTIL ROLL FORWARD RETRY
- RFUTIL AIMAGE AIOFF

- RFUTIL AIMAGE END
- RFUTIL AIMAGE SCAN
- RFUTIL AIMAGE EXTENT EMPTY
- RFUTIL AIMAGE TRUNCATE
- RFUTIL AIMAGE EXTENT FULL

8.5.3 System Crash While Backing Up the Database

If the system crashes while you are backing up the database, restart the backup.

8.5.4 System Crash While Database Is Up

If the system crashes while the database is up, restart the database. Crash recovery automatically performs, using the information stored in the BI files.

NOTE: If you run the database with the `-r` parameter and your system fails because of a system crash or power failure, you cannot recover the database. If you run the database with the `-i` parameter and it fails for any reason, you cannot recover the database.

8.6 Recovering From Media Failures

After-imaging combined with backup and roll-forward recovery lets you recover the database if a disk holding a database file is damaged. After-imaging is optional; you must enable it before a disk fails, and you must have performed backups to use roll-forward recovery. You should also keep backups and archived AI files until they are no longer required.

The technique for recovering the database depends on the circumstances that require recovery. The next sections provide scenarios and explanations of how to use the PROUTIL and RFUTIL utilities to handle the following problems:

- Loss of the DB files, BI files, or both
- Loss of the AI files
- Loss of the database backup
- Loss of the TL file

8.6.1 Loss Of the DB Files, BI Files, Or Both

If a disk holding the DB or BI files is damaged, or the files are accidentally deleted, follow these steps to restore the database:

1 ♦ Back up the current AI file and any full AI files not already archived before rolling forward. This step is important to protect yourself in case you lose the AI files if anything goes wrong, such as inadvertently overwriting the AI file. Be sure to back up the AI files on media different from the media where the database files are backed up. The AI files have the information you require to bring the database backup files up to date.

2 ♦ Create the structure for the database. You will restore the database to this structure.

3 ♦ Restore the most recent full database backup. If you are using incremental backups, you must restore the most recent full backup and then apply all the incremental backups.

4 ♦ If you use after-imaging, roll forward the AI files starting after the last backup. Use the RFUTIL ROLL FORWARD utility. See Chapter 11, “After-Imaging,” for more information.

5 ♦ Perform a full backup of the restored, recovered database. Follow the standard backup procedure described in Chapter 7, “Backing Up a Database.”

6 ♦ Restart after-imaging and two-phase commit, if you are using it.

7 ♦ Restart the database.

8 ♦ Restart the applications and continue database processing.

8.6.2 Loss Of the AI File

Follow these steps if a disk holding the AI files is damaged or the AI file is accidentally deleted:

1 ♦ If two-phase commit is enabled for the database, end two-phase commit using the PROUTIL 2PHASE END utility. You must do so before you can perform the next step.

NOTE: If you end two-phase commit and a coordinator recorded any distributed transaction commits in this database, the user cannot resolve any limbo transactions where this database was the coordinator.

2 ♦ Disable after-imaging using the RFUTIL AIMAGE END utility.

3 ♦ If you have a database, re-create the lost extent using PROSTRCT ADD.

4 ♦ Back up the database using the PROBKUP utility or an operating system backup utility.

5 ♦ If you used an operating system backup utility to back up the database, mark the database as backed up using the RFUTIL MARK BACKEDUP utility.

6 ♦ Enable after-imaging.

7 ♦ Re-enable two-phase commit if you disabled it in Step 1.

8.6.3 Loss Of Database Backup

If after-imaging is enabled for a database, you can recover from a lost or corrupted database backup as long as you have archived the AI files every time you backed up the database.

Follow these steps to restore the database:

1 ♦ Archive the current AI file to ensure that a second copy is available in case the original extents are damaged (for example, if you inadvertently overwrite the current busy AI file when restoring archived AI files in Step 3). The AI files have the information you need to bring the database backup files up to date.

   Be sure to archive the AI files on different media than the database files backup.

2 ♦ Restore the backup you made immediately before the lost backup. If you are using incremental backups, you must restore the most recent full backup and then apply all the incremental backups to the full backup.

3 ♦ Restore the AI files you archived at the same time you made the backup that was lost or corrupted.

4 ♦ Use the RFUTIL ROLL FORWARD utility to roll forward the AI files you restored in Step 3. See Chapter 11, “After-Imaging,” for more information.

5 ♦ Use the RFUTIL ROLL FORWARD utility to roll forward the current AI files (that is, the AI files in use since the lost backup).

6 ♦ Back up the database following the standard backup procedure described in Chapter 7, “Backing Up a Database.”

7 ♦ Enable after-imaging and two-phase commit if you are using them.

8 ♦ Restart the database.

9 ♦ Restart any applications and continue database processing.

8.6.4 Loss Of Transaction Log File

Follow these steps if you cannot access the database because of a lost or corrupt transaction log file:

1 ♦ If two-phase commit is enabled for the database, end two-phase commit using the PROUTIL 2PHASE END utility.

NOTE: If you end two-phase commit and a coordinator recorded any distributed transaction commits in this database, the user cannot resolve any limbo transactions for that distributed transaction unless after-imaging is also enabled for the database.

2 ♦ Enable two-phase commit using the PROUTIL 2PHASE BEGIN utility.

3 ♦ Put the database back online.

8.7 Recovering From a Full Disk

The database engine terminates if a disk holding a database file becomes full while the Progress database is running. If this happens, follow these steps:

1 ♦ Remove any nonessential files from the full disk.

2 ♦ Start the Progress database, letting it automatically perform crash recovery of the database.

If you cannot make enough room on your disk to allow the database engine to perform crash recovery, you must take further steps. The steps vary, depending on the contents of the full disk, as explained in the following sections.

8.7.1 After-image Area Disk

If a variable-length after-image extent fills a disk, the database engine automatically switches to the next AI extent if that extent is empty. If the extent is not empty, the database shuts down unless the After-image Stall (`-aistall`) startup parameter was specified.

Follow these steps if the database shuts down because the engine cannot switch to the next extent:

1 ♦ Back up the least recently used full after-image extent. Use the RFUTIL AIMAGE EXTENT FULL utility to find out which extent this is.

2 ♦ Use the RFUTIL AIMAGE EXTENT EMPTY utility to mark the extent as available for reuse.

3 ♦ Restart the database.

8.7.2 Control Or Primary Recovery Area Disk

The procedure you follow to recover from a full disk depends on whether the variable-length BI extent is on the full disk. Because the BI extent grows during the crash recovery phase of the Progress database startup, if the BI extent is on the full disk, you cannot start the database.

Variable-length BI Extent Is Not On the Full Disk

Follow these steps if the BI file is not on the full disk:

1 ♦ Truncate the BI file using the PROUTIL TRUNCATE BI utility. This rolls back any transactions active at the time the database shuts down. You can also start a single-user Progress session to accomplish the same thing.

2 ♦ Add one or more data extents to the database using the PROSTRCT ADD utility. This step is not necessary to bring the database back online. However, if you omit this step, the database will shut down when users begin adding data to it.

   If you have no additional disk space, you must install a new disk or delete some data from the database before you can bring the database online.

3 ♦ Restart the database.

Variable-length BI Extent Is On the Full Disk

The BI extent grows during the crash recovery phase of the Progress database startup. If the BI extent is on the full disk, you will not be able to start the database until you make space available for the BI file to expand.

Follow these steps if you cannot remove any files on the disk containing the variable-length BI extent:

1 ♦ Use the PROSTRCT ADD utility to add a BI extent on a disk that contains some free space. The maximum amount of additional BI space you require to recover the database is equal to that currently consumed by the BI file. If you have a disk with this much free space, then assign the new BI extent to that volume. Note that this is a worst-case space requirement and the BI file normally grows much less than this. Adding the BI extent makes the variable length extent a fixed-length extent.

2 ♦ Start a single-user Progress session against the database. This begins database crash recovery. Exit the single-user Progress session.

3 ♦ Restart the database.

8.7.3 Transaction Log File Disk

Follow these steps if two-phase commit is enabled:

1 ♦ Use PROUTIL 2PHASE END to disable two-phase commit on the database.

NOTE: Unless after-imaging is also enabled for the database when you end two-phase commit, the user will be unable to resolve any limbo transactions where the database recorded the coordinator's commit of a distributed transaction.

2 ♦ Truncate the BI file.

3 ♦ Enable two-phase commit.

4 ♦ Restart the database.

8.8 Truncating the BI File

Truncating the BI file resets the file to its minimum size. You should plan ahead to determine the amount of disk space required to hold all the files required to process at your peak transaction rate. However, under certain circumstances, the BI file might grow unusually large. This usually occurs because of a long-running transaction, for example, if a user leaves the terminal for a long period in the middle of a transaction. If a situation like this occurs, you can truncate the BI file.

When you truncate the BI file, the database engine uses the information in the BI files to bring the database and AI files up to date, waits to verify that the information has been successfully written to the disk, then truncates the primary recovery file to its minimum length.

To truncate the BI file, use the TRUNCATE BI qualifier of the PROUTIL utility:

```
proutil db-name -C truncate bi
```

In this command, *db-name* specifies the database you are using.

See the description of the PROUTIL utility in Chapter 19, "Database Administration Utilities," for a complete description of the TRUNCATE BI qualifier.

8.9 Recovering From a Crash

If the broker dies or is killed by some means other than the database shutdown command, it does not release attached shared-memory segments. Use PROUTIL -C DBIPCS to verify shared memory has not been released; use the UNIX command `ipcrm -m` to free it.

Use the DBIPCS qualifier on PROUTIL to display the status of shared-memory segments attached by all Progress databases on the system:

```
PROUTIL -C DBIPCS
```

You do not have to include a database name. Here is an example of the status output:

```
Progress SHARED MEMORY STATUS
 ID  ShMemVer  Seg#  InUse   Database
 68      -      -     -      (not Progress)
100      3      0     Yes    /db/work5/sports
101      3      1     -      /db/work5/sports
120      3      0     No     /db/work5/test
150      2      -     -      (unsupported shared memory version)
```

Table 8–6 describes the display fields in the output.

Table 8–6: Shared-memory Segment Status Fields

Field	Description
ID	Indicates the shared-memory ID.
ShMemVer	Specifies the shared-memory version.
Seg#	Indicates the shared-memory segment number. One database can own more than one segment.
InUse	Specifies whether the segment is in use. Yes or No values are displayed only if the segment is number 0. All other segments show a dash (-). To determine whether a set of segments is in use, check the InUse value of segment 0 for the relevant database.
Database	Represents the full path name of the database.

8.10 Recovering From a Lost Or Damaged Control Area

Use the PROSTRCT BUILDDB utility to recover when an existing database control area (DB) is lost or damaged. The utility re-creates a control area from the structure description (ST) file of an existing database. For example:

```
prostrct builddb db-name structure-description-file
```

PROSTRCT BUILDDB uses the following parameters:

db-name

Specifies the database you are using.

structure-description-file

Specifies the existing structure description (ST) file. If a structure description file is not specified, PROSTRCT BUILDDB assumes the name is *db-name*.st.

PROSTRCT BUILDDB does only minimal validation of the resulting control area.

8.11 Unlocking Damaged Databases

With databases, inconsistencies can occur between extents from accidental misuse of operating system copy utilities or from incorrectly administered backup and restore procedures. The database engine synchronizes the opening and updating of the DB, BI, and AI extents to ensure the consistency of the database. If the engine finds an inconsistency among these extents, it returns an error message and stops any attempt to open the database. If this happens, try to restore the database from backups. As a last resort, you can use the PROSTRCT utility with the UNLOCK qualifier to open the database to dump the data. Use PROSTRCT UNLOCK only as a last resort. The locked database might not unlock, or the resulting unlocked database might be damaged and invalid.

Follow these steps to unlock a damaged database:

1 ♦ Unlock the database by entering the following command:

```
prostrct unlock db-name
```

CAUTION: If you are missing any database extents when you run PROSTRCT UNLOCK, Progress automatically replaces any missing extents with empty formatted extents and displays a message. You can determine whether this has occurred the next time any utility or program tries to open the database.

2 ♦ Use the Data Administration tool if you are using a graphical interface or the Data Dictionary if you are using a character interface to dump the data definitions and the table contents.

3 ♦ Exit the database.

4 ♦ Create a new void database.

5 ♦ Restart the database and reload the dumped data definitions and data into the new database.

8.12 Dumping Tables From a Damaged Database

Occasionally, data is damaged and the index rebuild utility does not work. When this happens, you can use the following Progress 4GL procedure to dump the data in a table called item. The `p-dump.p` procedure runs slowly on large databases. Use it only when you cannot use the Index Rebuild utility:

p-dump.p

```
DEFINE VARIABLE i AS INTEGER.
FIND _file "item".
OUTPUT TO item.d.
DO i = 1 TO 10000:
  FIND item WHERE RECID(item) = i NO-ERROR.
  IF AVAILABLE item AND i <> INTEGER(_file._template)
  THEN EXPORT item.
END.
```

In this procedure, data from the item table is dumped into the `item.d` file. Use the `item.d` file with the reload procedure. For more information on loading, see Chapter 13, “Dumping and Loading.”

In the `p-dump.p` procedure, you must set the end value for the DO block high enough (10,000 in the previous example procedure) so that every record in the database is examined. Calculate a safe value using the following formula:

```
100 + 32 (database-size-in-bytes / database-block-size)
```

The database block size varies among systems. Use the PROSTRCT STATISTICS utility to determine the database block size for your system.

8.13 Forcing Access To a Damaged Database With the -F Parameter

If you are unable to regain access to your database, you should restore the database from backup. If you have no backup and no other alternative, you can use the -F startup parameter to force entry to a damaged database.

CAUTION: You should use the -F parameter only as a last resort, since it compromises the integrity of the database.

Follow these steps to force access to a damaged database:

1 ♦ Back up the database.

2 ♦ Execute the PROUTIL command using the -F parameter:

```
proutil db-name -C truncate bi -F
```

In this command, *db-name* specifies the name of the database.

The following messages appear:

```
The -F option has been specified to proutil. (6260)
Forcing into the database skips database recovery. (6261)
```

3 ♦ Start a single-user Progress session against the database.

4 ♦ Use any of the database administration tools or utilities to dump and then load the database. See Chapter 13, "Dumping and Loading," for more information.

CAUTION: Forcing access to a damaged database is an action of last resort and might not result in successfully opening the database. If the database cannot open, none of the Progress recovery tools can be used to dump the contents of the database.

9

Maintaining Database Structure

Once you create and start a database, you must manage the database so that it operates efficiently and meets the needs of users. This chapter describes methods to manage the database structure and alter it as necessary to improve storage and performance. It contains the following sections:

- The Progress Structure Utility
- Progress Structure Statistics Utility
- Progress Structure List Utility
- Progress Structure Add Utility
- Progress Structure Remove Utility
- Maintaining Indexes and Tables
- Using Virtual System Tables To Obtain Status Of Administration Utilities

9.1 The Progress Structure Utility

Using the Progress Structure (PROSTRCT) Utilities, you can scale your Version 9 database to meet business requirements. For example, if your business requires high availability, use PROSTRCT to reorganize tables and indexes while users operate against the database. The following sections detail how to maintain the structure of your Version 9 database, including when to use PROSTRCT and PROUTIL utilities.

9.2 Progress Structure Statistics Utility

Use the Progress Structure Statistics Utility (PROSTRCT STATISTICS) to monitor database growth and the availability of free blocks for storing additional data. For example:

```
prostrct statistics db-name
```

You specify the database whose storage information you want and the PROSTRCT STATISTICS utility displays information about:

- The database name
- The primary database block size and the before-image and after-image block sizes
- The storage area and each extent within it
- The total number of active blocks allocated for each data area
- The total number of empty blocks in each data area
- The total number of extent blocks for each data area
- The total number of blocks for each data area
- The total records per block for each data area
- The total number of all blocks (active, data, free, empty, extent, and total blocks)
- The date and time of the last full database backup

For example, this command displays database storage information for the database /usr/joe/service.db:

```
prostrct statistics /usr/joe/service.db
```

If a full backup has not yet been performed against service.db, the message would read "NO FULL BACKUP HAS BEEN DONE."

9.3 Progress Structure List Utility

Your structure description file must reflect the current information in your database control area to ensure continuity and ease database management. Therefore, update the structure description file any time you make changes to the structure of a database, such as adding, moving, or removing extents. Use the Progress Structure List (PROSTRCT LIST) utility to update your structure description file. PROSTRCT LIST produces a new structure description file for the database you name, using the current information stored in the database control area.

NOTE: You can use PROSTRCT LIST with an online database.

To update the structure description file with the current information stored in the database control area, use the PROSTRCT LIST utility:

```
prostrct list db-name [ structure-description-file ]
```

db-name

Specifies the database whose structure description file you want to update.

structure-description-file

Specifies the structure description file to create. If you do not specify the structure description file, PROSTRCT LIST uses the base name of the database and appends a .st extension. It replaces an existing file of the same name.

For example, to update the structure description file for /user/joe/service, enter the following command:

```
prostrct list /user/joe/service
```

PROSTRCT LIST displays the information it produces in the structure description file, including storage area type, storage area name, records per block, extent pathname, and extent type, either fixed or variable (a size of 0 indicates a variable-length extent). For example:

```
AreaName: ControlArea, Type 6, Block Size 1024, Extents 1, Records/Blck 32
   Ext # 1, Type VARIABLE, Size 0, Name: /user/joe/service.db

AreaName: Primary Recovery Area, Type 3, Block Size 8192, Extents 1
   Ext # 1, Type VARIABLE, Size 0, Name: /usrer/joe/service.b1

AreaName: Transaction Log Area, Type 4, Block Size 16384, Extents 1
   Ext # 1, Type FIXED   , Size 1024, Name:/user/joe/service.t1

AreaName: Schema Area, Type 6, Block Size 1024, Extents 1, Records/Blck 32
   Ext # 1, Type VARIABLE, Size 0, Name: /user/joe/service.d1

AreaName: After Image Area 1, Type 7, Block Size 8192, Extents 1
   Ext # 1, Type FIXED   , Size 32, Name: /user/joe/service.a1

AreaName: service, Type 6, Blck Size 1024, Extents 6, Records/Block 32
   Ext # 1, Type FIXED   , Size 32, Name:/user/joe/service_8.d1
   Ext # 2, Type FIXED   , Size 32, Name:/user/joe/service_8.d2
   Ext # 3, Type FIXED   , Size 32, Name:/user/joe/service_8.d3
   Ext # 4, Type FIXED   , Size 32, Name:/user/joe/service_8.d4
   Ext # 5, Type FIXED   , Size 32, Name:/user/joe/service_8.d5
   Ext # 6, Type VARIABLE , Size 0, Name:/user/joe/service_8.d6

AreaName: joe, Type 6, Block Size 1024, Extents 1, Records/Block 32
   Ext # 1, Type VARIABLE, Size 0, Name: /user/joe/yy/z/service_1000.d1
```

9.4 Progress Structure Add Utility

Use the Progress Structure Add (PROSTRCT ADD) utility to add storage areas and extents to an existing database. Note that the database must be offline before you can use PROSTRCT ADD. To add storage areas to an existing database using PROSTRCT ADD, follow these steps:

1 ♦ Back up your database.

2 ♦ Create a new structure description (ST) file that contains only information about the areas you want to add. For example:

```
# add.st
#
d "chris",128 . f 1024
d "chris" .
```

NOTE: To avoid overwriting the ST file for your existing database, the name of this ST file must be different from the existing ST file for the database. For example, name the new structure description file `add.st`.

3 ♦ Use the PROSTRCT Utility with the ADD qualifier, specifying the ST file you created in Step 2. For example:

```
prostrct add db-name incremental-description-file
```

PROSTRCT ADD adds the new extents or storage areas and extents to the existing database control area, and outputs descriptive information such as:

```
Formatting extents:
   size              area name           path name
   1024                chris        /user/joe/service_9.d1  00:00:00
     32                chris        /user/joe/service_9.d2  00:00:01
```

NOTE: After you modify the structure of your database (adding or removing extents), run PROSTRCT LIST. PROSTRCT LIST automatically creates a structure description (ST) file for your database if one does not exist, or overwrites an existing ST file of the same name to reflect the changes you just made to the structure of your database.

9.4.1 PROSTRCT ADD and Pathnames

When you use PROSTRCT ADD to add additional storage areas or extents to a relative path database, the relative path database is converted to an absolute path database. For example, if you add an additional application data storage area named `Misc` to the relative path database `example1`, PROSTRCT ADD converts the relative path database to an absolute path database. Use PROSTRCT LIST to update the ST file with the added extents. For example:

```
prostrct add example1 add.st
prostrct list example1
```

In the following sample output of the PROSTRCT LIST utility, note the absolute pathname (`/usr1/example1.db`) of the modified database:

```
Area Name: Control Area, Type6, BlockSize 1024, Extents 1, Records/Block32
   Ext # 1, Type VARIABLE, Size 0, Name: /usr1/example1.db

Area Name: Primary Recovery Area, Type3, BlockSize 8192, Extents 1
   Ext # 1, Type VARIABLE, Size 0, Name: /usr1/example1.b1

Area Name: Schema Area, Type6, BlockSize 1024, Extents 1, Records/Block 32
   Ext # 1, Type VARIABLE, Size 0, Name: /usr1/example1.d1

Area Name: Employee, Type6, BlockSize 1024, Extents 2, Records/Block 32
   Ext # 1, Type FIXED   , Size 320, Name: /usr1/example1_7.d1
   Ext # 2, Type VARIABLE, Size 0, Name: /usr1/example1_7.d2

Area Name: Inventory, Type6, BlockSize 1024, Extents 2, Records/Block 32
   Ext # 1, Type FIXED   , Size 608, Name: /usr1/cmcguire/example1_8.d1
   Ext # 2, Type VARIABLE, Size 0, Name: /usr1/example1_8.d2

Area Name: Cust_Data, Type6, BlockSize 1024, Extents 2, Records/Block 32
   Ext # 1, Type FIXED   , Size 320, Name: /usr1/example1_9.d1
   Ext # 2, Type VARIABLE, Size 0, Name: /usr1/example1_9.d2

Area Name: Cust_Index, Type 6, BlockSize 1024, Extents 2, Records/Block 32
   Ext # 1, Type FIXED   , Size 320, Name: /usr1/example1_10.d1
   Ext # 2, Type VARIABLE, Size 0, Name: /usr1/example1_10.d2

Area Name: Order, Type6, BlockSize 1024, Extents 2, Records/Block 32
   Ext # 1, Type FIXED   , Size 1280, Name: /usr1/example1_11.d1
   Ext # 2, Type VARIABLE, Size 0, Name: /usr1/example1_11.d2

Area Name: Misc, Type6, BlockSize 1024, Extents 2, Records/Block 32
   Ext # 1, Type FIXED   , Size 320, Name: /usr1/example1_12.d1
   Ext # 2, Type VARIABLE, Size 0, Name: /usr1/example1_12.d2
```

9.4.2 Adding Extents to Existing Storage Areas

To add extents to an existing area use the PROSTRCT utility with the ADD qualifier.

NOTE: You can only add after-image (AI) extents when after-imaging is disabled. You can only add transaction-log (TL) extents when the database is offline.

To add extents to an existing database using PROSTRCT ADD, follow these steps:

1 ♦ Back up your database.

2 ♦ Create a new structure description (ST) file that contains only information about the areas you want to add. For example:

```
# add.st
#
d "chris",128 .
```

NOTE: To avoid overwriting the ST file for your existing database, the name of this ST file must be different from the existing ST file for the database. For example, name the new structure description file `add.st`.

3 ♦ Use the PROSTRCT Utility with the ADD qualifier, specifying the ST file you created in Step 2. For example:

```
prostrct add db-name incremental-description-file
```

PROSTRCT ADD adds the new extents or storage areas and extents to the existing database control area and outputs descriptive information such as:

```
Formatting extents:
   size            area name         path name
     0               chris        /user/joe/service_9.d3  00:00:01
```

NOTE: After you modify the structure of your database (such as adding or removing extents), run PROSTRCT LIST. It automatically creates a structure description (ST) file for your database if one does not exist, or overwrites an existing ST file of the same name to reflect the changes you just made to the structure of your database.

9.5 Progress Structure Remove Utility

Use the Progress Structure Remove (PROSTRCT REMOVE) to remove extents from storage areas. After all the extents are removed from a storage area, PROSTRCT REMOVE removes the storage area. Follow these steps to remove storage areas:

1 ♦ If the extent to be removed is in the BI area, use the PROUTIL TRUNCATE BI utility to truncate the primary recovery area to be removed. For example:

```
proutil db-name -C truncate bi
```

If the storage area to be removed is an application data area, remove all tables and indexes, then use the PROUTIL TRUNCATE AREA utility to truncate the application data area. For example:

```
proutil db-name -C truncate area area-name
```

NOTE: You must disable after-imaging before you can remove an AI extent. You must also disable two-phase commit before you can remove a transaction-log (TL) extent.

For more information about truncating areas, see the "PROUTIL TRUNCATE AREA Qualifier" section in Chapter 19, "Database Administration Utilities."

2 ♦ Use PROSTRCT REMOVE to remove extents from the storage area. For example:

```
prostrct remove db-name extent-type-token [area-name]
```

NOTE: Use the *area-name* parameter only to remove application data extents. If the area name contains a space, supply double-quotes around the area name. For example: `"test data."`

You can remove one extent at a time. After you have removed all of the extents from a storage area, PROSTRCT REMOVE removes the storage area and outputs descriptive information such as:

```
solaris:90a$ prostrct remove service d chris
/user/joe/service_9.d3 successfully removed
solaris:90a$ prostrct remove service d chris
/user/joe/service_9.d2 successfully removed
solaris:90a$ prostrct remove service d chris
/user/joe/service_9.d1 successfully removed
```

3 ♦ Run PROSTRCT LIST after removing any areas. PROSTRCT LIST will overwrite your existing ST file to reflect the changes made to the structure of your database.

9.6 Maintaining Indexes and Tables

You can add tables and indexes to existing storage areas in a database using the Create Table and Create Index commands from the Progress 4GL Data Dictionary. However, you cannot add tables or indexes to existing storage areas using the Modify Table command or the Index property sheet. The following sections detail how to move tables and indexes and how to compact indexes.

9.6.1 Moving Tables

Use the PROUTIL TABLEMOVE utility to move a table and its associated indexes from one storage area to another while the database remains online. For example:

```
proutil db-name -C tablemove [owner-name.]table-name table-area
[ index-area ]
```

NOTE: For the complete syntax description, see Chapter 19, “Database Administration Utilities.”

If you omit the *index-area* parameter the indexes associated with the table will not be moved.

Moving the records of a table from one area to another invalidates all the ROWIDs and indexes of the table. Therefore the indexes are rebuilt automatically by the utility whether you move them or not. You can move the indexes to an application data area other than the one to which you are moving the table. If you want to move only the indexes of a table to a separate application data area, use the PROUTIL IDXMOVE utility.

Moving a table's indexes with the TABLEMOVE qualifier is more efficient than moving a table separately and then moving the indexes with the IDXMOVE utility. Moving a table separately from its indexes wastes more disk space and causes the indexes to be rebuilt twice, which also takes longer.

NOTE: While you can move tables online, no access to the table or its indexes is recommended during the move. The utility acquires an EXCLUSIVE lock on the table while it is in the process of moving. An application that reads the table with an explicit NO-LOCK might be able to read records, but in some cases can get the wrong results, since the table move operation makes many changes to the indexes. Progress Software Corporation recommends that you run the utility during a period when the system is relatively idle, or when users are doing work that does not access the table.

The PROUTIL TABLEMOVE utility operates in phases:

- Phase 1: The records are moved to the new area and a new primary key is built.
- Phase 2: All the secondary indexes are built.
  - If you did not specify the *index-area* parameter then the indexes are rebuilt in their original area.
  - If you did specify the *index-area* parameter then all the indexes are moved to the new area where they are rebuilt.
- Phase 3: All the records in the old area are removed.
- Phase 4: All the old indexes are removed and the _StorageObject records of the indexes and the table are updated.

NOTE: Although PROUTIL TABLEMOVE operates in phases, it moves a table and its indexes in a single transaction. To allow a full recovery to occur when a transaction is interrupted, every move and delete of each individual record is logged. As a result, moving a table requires the BI Recovery Area to be several times larger than the combined size of the table and its indexes. Therefore, before moving your table, determine if your available disk capacity is sufficient to support a variable BI extent that might grow to more than three times the size of the table and its indexes.

The _UserStatus virtual system table displays the utility's progress. For more information see Chapter 20, "Virtual System Tables."

9.6.2 Moving Indexes

Use the PROUTIL IDXMOVE utility to move an index from one application data area to another while the database remains online. You might be able to improve performance by moving indexes that are heavily used to an application data area on a faster disk. For example:

```
proutil db-name -C idxmove [owner-name.]indexname area-name
```

NOTE: For the complete syntax description, see Chapter 19, "Database Administration Utilities."

The PROUTIL IDXMOVE utility operates in two phases:

- Phase 1: The new index is being constructed in the new area. The old index remains in the old area and all users can continue to use the index for read operations.
- Phase 2: The old index is being removed and all the blocks of the old index are being removed to the free block chain. For a large index this phase can take a significant amount of time. During this phase all operations on the index are blocked until the new index is available; users accessing the index might experience a freeze in their application.

The _UserStatus virtual system table displays the utility's progress.

NOTE: While you can move indexes online, no writes to the table or its indexes are allowed during the move. The IDXMOVE utility acquires a SHARE lock on the table, which blocks all attempts to modify records in the table. Progress Software Corporation recommends that you run the utility during a period when the system is relatively idle, or when users are doing work that does not access the table.

9.6.3 Compacting Indexes

When the DBANALYS utility indicates that space utilization of an index is reduced to 60% or less, use the PROUTIL IDXCOMPACT utility to perform index compaction online. Performing index compaction increases space utilization of the index block to the compacting percentage specified. For example:

```
proutil db-name -C idxcompact [ owner-name.]table-name.index-name [ n ]
```

NOTE: For the complete syntax description, see Chapter 19, "Database Administration Utilities."

Performing index compaction reduces the number of blocks in the B-tree and possibly the number of B-tree levels, which improves query performance.

The index compacting utility operates in phases:

- Phase 1: If the index is a unique index, the delete chain is scanned and the index blocks are cleaned up by removing deleted entries.
- Phase 2: The nonleaf levels of the B-tree are compacted starting at the root working toward the leaf level.
- Phase 3: The leaf level is compacted.

The _UserStatus virtual system table displays the utility's progress.

NOTE: Because index compacting is performed online other users can use the index simultaneously for read or write operation with no restrictions. Index compacting only locks 1 to 3 index blocks at a time, for a short time. This allows full concurrency.

9.7 Using Virtual System Tables To Obtain Status Of Administration Utilities

Depending on the size and complexity of your database, some online database administration utilities can take a significant amount of time to complete. You can use the _UserStatus virtual system table (VST) to monitor the state of the following administration utilities that can be performed online:

- PROUTIL IDXANALYS – displays information about index blocks
- PROUTIL IDXCOMPACT – increases space utilization of an index block
- PROUTIL IDXFIX – detects corrupt indexes and records with a missing or incorrect index
- PROUTIL IDXMOVE – moves indexes among application data storage areas
- PROUTIL TABLEMOVE – moves tables among application data storage areas

For more information on VSTs, see Chapter 20, "Virtual System Tables."

EXAMPLES

Using the virtual system table mechanism, you can monitor the status and progress of several database administration utilities.

The following example describes how to use a Progress 4GL routine to monitor all the index move (PROUTIL IDXMOVE) processes being performed:

```
/* Monitor all "index move" processes every three seconds */
REPEAT:
  FOR EACH _UserStatus WHERE _UserStatus-Operation = "Index move":
    DISPLAY _UserStatus.
  END.
PAUSE 3.
END.
```

The following is an equivalent SQL-92 statement to monitor all the index move (PROUTIL IDXMOVE) processes being performed:

```
    SELECT * FROM _UserStatus WHERE _UserStatus-Operation = "Index move";
```

The following example describes how to use a Progress 4GL routine to monitor the status of all administration utilities performed:

```
/*Monitor all user processes who report their status (admin utilities)*/
REPEAT:
  FOR EACH _UserStatus WHERE _UserStatus-Operation <> ?:
    DISPLAY _UserStatus.
  END.
PAUSE 3.
END.
```

10

Maintaining Security

As a Progress database administrator, you want to ensure that only authorized users connect to a database, as well as prevent unauthorized users from modifying or removing database files and objects. This chapter describes how to maintain the security of your Progress database. It contains the following sections:

- Establishing a Progress User ID and Password
- Establishing Authentication For Your Progress Database
- Connection Security
- Running a User Report
- Schema Security
- Operating Systems and Database Security

NOTE: For more information on compile-time and run-time security, see the chapter on providing data security in the *Progress Programming Handbook*.

CAUTION: Before changing your database's security implementation, make a backup of the original database. This way you can restore the original database if you cannot undo some of the security measures you set. For example, if you are the sole security administrator and you forget your user ID or password, you might be denied access to the database and the data contained in it. However, if you have a backup of the original database, you can restore it. To prevent such a mishap, Progress Software Corporation recommends that you designate at least two security administrators for each database.

10.1 Establishing a Progress User ID and Password

Every Progress database connection has an associated user ID. Each type of Progress application security uses the user ID to determine the user's authority to access data.

At the first level, connection security, you decide whether to allow all users the authority to connect to a database or to establish a list of authorized user IDs. Each user ID can be further protected by a password to prevent unauthorized users from connecting to the database.

If you choose **not** to establish a list of valid user IDs, then the database inherits user IDs or user names from the operating system.

On UNIX, the database uses the operating system user ID. Note that the user IDs or user names on these operating systems must adhere to Progress rules on user IDs. For more information, see the "Progress User ID" section.

If you choose to establish a list of valid user IDs with passwords, then you must enter the data into the *user list*. You can access the user list through the Data Dictionary Security menu. The database engine stores the user list in the _User table, which is hidden from users. From then on, the user must specify a user ID and password when connecting to the database. Note that the database no longer inherits the user ID or user name from the operating system once you have established this list.

In an interactive session, the application can prompt the user for a user ID and password. In a batch session, the user can specify a user ID and password by using the User ID (`-U`) and Password (`-P`) startup parameters.

10.1.1 Progress User ID

A *user ID* is a string of up to 12 characters associated with a particular Progress database connection. User IDs can consist of any printable character or digit except the following: #, *, !, @. User IDs are not case sensitive; they can be uppercase, lowercase, or any combination of these. A user ID can be blank, written as the string " ", but you cannot define it as such through the Data Dictionary.

10.1.2 Progress Password

A *password* is a string of up to 16 characters that is associated with a user ID. When you add the password to the user list, the password is encoded with the ENCODE function. Because ENCODE returns different values for uppercase and lowercase input, all Progress passwords are case sensitive.

10.1.3 Validating a Progress User ID and Password

If you establish a list of valid user IDs, Progress prompts the user for a user ID and password at connection time. Typically, the application does this by running the Progress login procedure. The standard Progress startup procedure, PROSTART, automatically runs a login procedure for each connected database. If the application uses another startup procedure, the developer should run a login procedure from that procedure.

The login procedure uses the SETUSERID function to check the user ID and password entered by the user. The user has three attempts to enter the correct user ID and password for each database. If the user fails to do so after three attempts, Progress exits the user from the database. If the user ID and password combination is valid for the database, then SETUSERID establishes that user ID for the connection.

If the application does not run the login procedure at connection time, or if the user bypasses the login procedure (by pressing **END-ERROR** when prompted for the user ID and password), then the user is assigned the blank user ID. While you cannot prevent users from connecting to the database with the blank user ID, you can prevent them from accessing data by establishing compile-time and run-time security.

10.2 Establishing Authentication For Your Progress Database

To prevent unauthorized users from accessing, modifying, or removing database files, you can establish appropriate database administration (DBA) and user-access privileges. The process for establishing authentication depends on whether your database has 4GL tables only, SQL-92 tables only, or both 4GL and SQL-92 tables

10.2.1 4GL Tables Only

By default, no user ID or password is required. You can use the Data Dictionary to designate a database administrator (DBA) user ID and use it to grant user access privileges. A Progress 4GL DBA has read, write, create, and delete permissions for the _USER table.

10.2.2 SQL-92 Tables Only

An SQL-92 database administrator (DBA) is a person assigned a `sysdbauth` record in the database. SQL-92 DBAs have access to all meta data and data in the database. To support Progress internal schema caching, every Progress database begins with a DBA defined as “sysprogress.” Progress restricts the use of “sysprogress,” however, and the process you follow to establish authentication depends on whether the database is newly created or created prior to Progress Version 9.0B.

In New Databases

When you create a new Progress database using the PROCOPY or PRODB commands, and the database does not have any _USER records defined (from the source database, for example), then a DBA is automatically designated with the login ID of the person who creates the database. This person can log into the database and use the GRANT statement to designate additional SQL-92 DBAs, and use the CREATE USER and DROP USER statements to add and delete user IDs.

In Earlier Databases

If your database was created earlier than Progress Version 9.0B and does not have any _USER records defined, then do the following:

1 ♦ Log in as "sysprogress."

2 ♦ Designate another DBA user ID and use it to grant user-access privileges.

NOTE: Use the GRANT statement to designate other SQL-92 DBAs. For extra authentication protection through password checking, use the CREATE USER and DROP USER statements. For more information about the GRANT, CREATE USER, and DROP USER statements, see the *Progress SQL-92 Guide and Reference*.

10.2.3 Both 4GL and SQL-92 Tables

When your database contains both 4GL and SQL-92 tables, the process you follow to establish authentication depends on whether the database is a new Progress database or a database created earlier than Progress Version 9.0B, with and without _USER records defined.

In New Databases

When you create a new Progress database using the PROCOPY or PRODB commands, your login ID is used to automatically designate you as DBA, but only if there are no predefined_USER records or DBAs (other than "sysprogress") in the source database.

Then, as creator of the new database, you can:

- Use the Data Dictionary to designate a 4GL DBA user ID and use it to grant access privileges.
- Log into the database and use the GRANT statement to designate additional SQL-92 DBAs, and use the CREATE USER and DROP USER statements to add and delete user IDs.

In Earlier Databases Without User Records Defined

If your database was created earlier than Progress Version 9.0B and does not have any _USER records defined, then do the following:

1 ♦ Log in as "sysprogress."

2 ♦ Use the GRANT statement to designate another SQL-92 DBA and to grant user-access privileges.

In Earlier Databases With User Records Defined

If your database was created earlier than Progress Version 9.0B and has _USER records defined, you must use the 4GL DBA permissions to create a "sysprogress" user ID and password. Only then can you access the database from SQL-92 and use the GRANT statement to create SQL-92 DBAs. The SQL-92 DBAs must have corresponding _USER records for them to log in to the database successfully.

NOTE: Once you add one _USER record to a Progress database, a user ID and password is then required by anyone who wants to access the database. Consequently, first create all DBAs and be sure that at least one has a _USER record, then add additional user IDs.

10.3 Connection Security

Connection security ensures that only authorized users connect to a database. To establish connection security, you use the Edit User List utility in the Data Dictionary to create a list of valid users. Once the user list is established, you can use the Security Administrators utility to appoint one or more security administrators. From then on, only security administrators can access the Edit User List utility.

Typically, users who are not security administrators have limited access to the Security menu. They can only modify their own passwords and run a quick user report.

10.3.1 Designating a Security Administrator

If you establish a list of valid user IDs, you must designate one user, or preferably more than one user, as security administrator. The security administrator is responsible for maintaining the security information in the database. Typically, the database administrator also serves as security administrator. Progress allows you to designate security administrators only if you have connected to the database with a nonblank user ID and if your user ID is already in the user list. You can then designate yourself, and preferably another user, as security administrators.

Once you designate security administrators, only they can access the following Data Dictionary utilities on the Security menu:

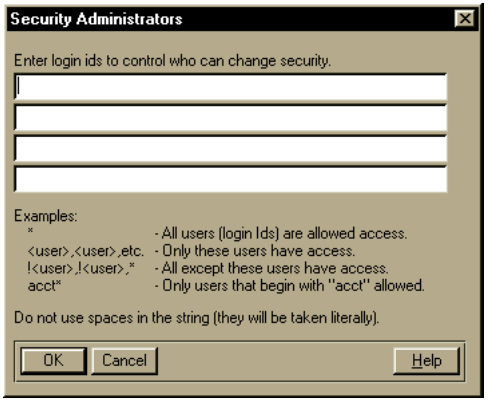
- Edit User List
- Change/Display Data Security
- Security Administrators
- Disallow Blank Userid Access

All users can access the other two utilities in the Security submenu: Change Your Password and Quick User Reports. Note that designating security administrators does not limit other users' ability to create new tables or fields in the database.

You can designate security administrators by using one of three utilities: Edit User List (CallAdmin option), Change/Display Data Security (CallAdmin option), and Security Administrators. These utilities allow you to access the same screen, where you follow these steps to enter user IDs for security administrators:

1 ♦ Access the Data Administration tool if you are using a graphical interface or access the Data Dictionary if you are using a character interface. The following steps represent the graphical interface.

2 ♦ Choose Admin→ Security→ Security Administrators. The Security Administrators dialog box appears:



3 ♦ You can enter many user IDs here, but you must include your own user ID. Otherwise, the following error message appears:

```
You cannot change a security field to exclude yourself.
```

Use commas, not spaces, to separate user IDs. If you use spaces in the string, they will be accepted as part of the string.

4 ♦ When you are done entering user IDs, choose OK. The Data Dictionary prompts you to verify the security administrator entries.

Verify that you want to save the entries. The Data Dictionary stores these user IDs as security administrators for the database.

10.3.2 Adding a User

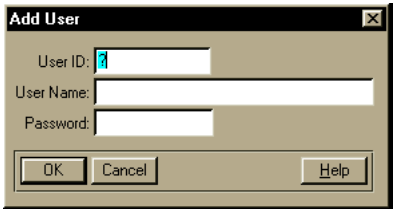
Follow these steps to add a user to the user list:

1 ♦ Access the Data Administration tool if you are using a graphical interface or access the Data Dictionary if you are using a character interface. The following steps represent the graphical interface.

2 ♦ Choose Admin→ Security→ Edit User List. The Edit User List dialog box appears:



3 ♦ Choose the Add button. The Add User dialog box appears:



Before you enter any value, remember that user IDs and user names are not case sensitive, but passwords are case sensitive. Anytime you want to cancel your entries, press **END-ERROR**.

4 ♦ Enter the user ID.

5 ♦ Enter the user name. The User Name field allows you to keep track of the user assigned to that user ID. Since Progress does not use the value in this field for security purposes, you can enter any text in this field.

6 ♦ Enter the password and choose OK. Progress prompts you to verify the password.

7 ♦ Enter the password again and choose OK. If you successfully enter the same password, you will see the user record added to the user list. If you enter a different password, no user record is created.

8 ♦ To add another user record, choose Add again. A new set of fields appears on the screen.

NOTE: You cannot change a User ID field or Password field once you have pressed **TAB**. However, you can change a user ID or password by deleting and then re-creating the user record. Note that the Modify button only allows you to make changes to the User Name field.

9 ♦ When you are done adding user records, choose OK.

After a user record is added to the user list, the user can connect to that database using the password you assigned. Users can then change the assigned passwords to their own private passwords by using the Change Your Password option on the Security menu.

If users forget their passwords, you can delete their records from the user list and re-create new user records for them.

10.3.3 Deleting a User

Follow these steps to delete a user from the user list:

1 ♦ Access the Data Administration tool if you are using a graphical interface or access the Data Dictionary if you are using a character interface.

2 ♦ Choose Admin→ Security→ Edit User List. The Edit User List dialog box appears.

3 ♦ Select the user you want to delete, then choose Delete. Progress prompts you to verify that you want to remove that user record. If you attempt to delete your own record, Progress does not allow you to do so until all existing user records are deleted.

4 ♦ Verify that you want to delete the user record.

5 ♦ When you are done deleting user records, choose OK.

If you delete all user records from the user list, Progress prompts you to confirm that you want to remove all security restrictions for the database. If you verify the deletions, Progress allows all users to have security administrator privileges when they connect to the database. If you choose No, you must add one or more users to the user list before you can exit to the Data Dictionary main window.

10.3.4 Changing a Password

This section describes how to change a password previously entered in the user list. Users can change their own passwords; they do not need security administrator privileges.

After you connect to the database with your user ID and password, follow these steps to change your password:

1 ♦ Access the Data Administration tool if you are using a graphical interface or access the Data Dictionary if you are using a character interface.

2 ♦ Choose Admin→ Security→ Change Your Password. Progress prompts you to enter your password.

3 ♦ Enter your password and choose OK. Remember that passwords are case sensitive. Progress prompts you to verify the new password.

4 ♦ Verify the password and choose OK.

As security administrator, you can change a user's user ID or password, but only do so by deleting and then re-creating the user record. This way, users cannot be locked out of a database if they forget their user IDs or passwords.

Do not try to bypass the Data Dictionary or Data Administration tool to modify passwords. You might lock yourself out of the database.

10.4 Running a User Report

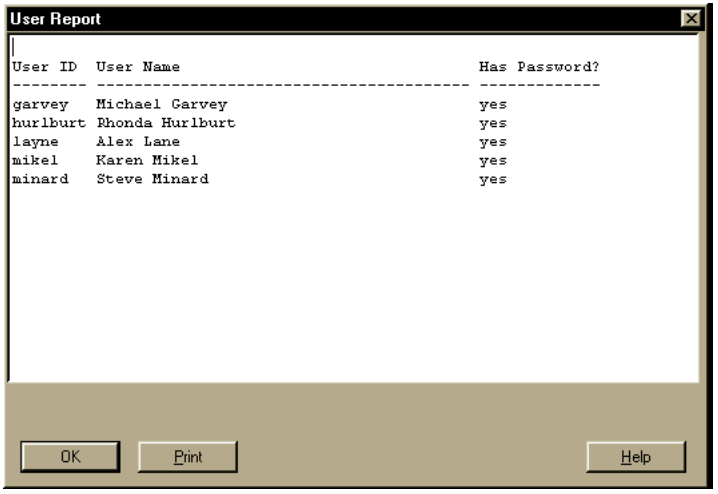
This section describes how to display or print a list of user IDs and user names, as well as how to find out which user IDs are protected by passwords.

All users can display and print a user report using the Quick User Report utility from the Security menu. In addition to this utility, security administrators can access three other utilities to run a user report: Edit User List, Change/Display Data Security, and Security Administrators.

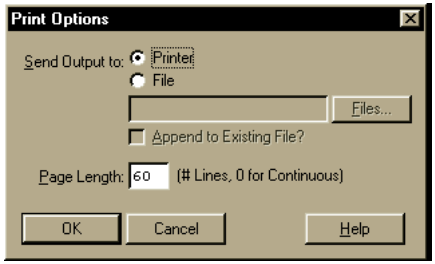
Follow these steps to generate a user report:

1 ♦ Access the Data Administration tool if you are using a graphical interface or access the Data Dictionary if you are using a character interface.

2 ♦ Choose Admin→ Security→ User Report or Database→ Reports→ User. Progress generates a user report similar to this:



3 ♦ Choose the Print button to output the report to a printer or file:



4 ♦ Specify the report destination. You can display the report on your terminal, print it, or save it to a file. If you want to save the report to a file, specify the filename and whether to append the report to an existing file.

5 ♦ Specify the page length and whether to order the fields.

6 ♦ Specify whether you want the report to include only the selected table or all the tables defined for the working database, then choose OK. If you specify a file or printer, Progress generates the report to the specified output destination and returns you to the Data Dictionary main window.

If you send the report to the terminal, the Detailed Table Report dialog box appears. Choose either the Change File Order button to reorder the fields or the Print button to print the report.

10.5 Schema Security

Schema security ensures that only authorized users can modify table, field, and index definitions. To establish schema security, use the Data Dictionary Freeze/Unfreeze utility to lock or freeze table, field, and index definitions in the database so that unauthorized users cannot make any modifications to them. Schema modifications change file time stamps, or cyclic redundancy check (CRC) values, making all procedures that reference the database invalid. After you freeze a database, no user can make any changes to it.

You can unfreeze frozen tables when you have to make changes, such as adding or deleting fields or indexes or modifying field or index definitions.

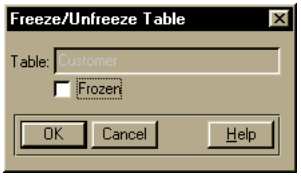
10.5.1 Freezing and Unfreezing a Table

Follow these steps to freeze or unfreeze a table:

1 ♦ Access the Data Administration tool if you are using a graphical interface or access the Data Dictionary if you are using a character interface.

2 ♦ Choose Utilities→ Freeze/Unfreeze. Progress alphabetically lists the tables defined for the working database.

3 ♦ Choose the table you want to freeze or unfreeze. The Freeze/Unfreeze Table dialog box appears:



4 ♦ Specify the table status, then choose OK.

As security administrator, you can control which users have authority to freeze or unfreeze database tables. To freeze or unfreeze tables, the user must have can-write access to the _File table and can-write access to the _File_Frozen field.

10.6 Operating Systems and Database Security

Each operating system provides security measures that you can use to protect your database.

You might want to allow access to the database from within a Progress 4GL session, but not from the operating system command level. On UNIX, you can use the operating system permissions to define security for the application database file.

Protecting the application database file using operating system permissions prevents users from accessing the database outside of the Progress 4GL, or with CONNECT statements from a Progress 4GL session, but it does not prevent anyone from starting single-user or multi-user Progress 4GL sessions. It also does not prevent anyone from accessing the database through a server.

To protect the database file with operating system permissions, create the database under a particular user ID. Change the permission of the database table by entering the following UNIX command:

```
chmod 600 db-name
```

This permission ensures that only the owner of the database file can access the file directly.

11

After-Imaging

The after-imaging feature lets you recover a database that was damaged when a failure caused the loss of the database or primary recovery (before image) area. When you enable after-imaging, the database engine writes notes containing a description of all database changes to the after-image (AI) files. You can use the AI files with the roll-forward recovery process to restore the database to the condition it was in before you lost the database, without losing completed transactions that occurred since the last backup. Chapter 8, "Recovering a Database," explains recovery mechanisms for Version 9 Progress databases, including after-imaging. This chapter details how to use after-imaging and contains the following sections:

- After-image Areas and Extents
- Estimating After-imaging Disk Space Requirements
- Creating After-image Areas
- Enabling After-imaging
- Managing After-imaging Files
- Performing Roll-forward Recovery
- Disabling After-imaging

11.1 After-image Areas and Extents

You can define multiple AI areas in the structure description (ST) file, which lets you create AI areas on multiple disk volumes and perform online backups while AI is enabled. Regardless of how many AI areas are defined, each AI area contains only one extent. Before defining the AI extents, consider the following:

- The database engine fills the AI areas in the order that you define them in the structure description file. When you are defining areas, you can store more than one AI area on a disk. However, you should store all the AI areas on disks other than the one that contains the database (DB) files or primary recovery area (BI) files.
- For both fixed-length and variable-length extents, the database engine automatically switches extents when the current extent becomes full, as long as the next extent is empty. If you define three large fixed-length extents, you can use extent 1 for a full day's worth of transactions, and have extent 2 empty and ready to use when you need to switch over to the next extent. This also leaves extent 3 available if you perform an unusually high number of transactions and use both extents 1 and 2.

The database engine uses AI areas sequentially, in the order defined in the structure description file. AI area filenames have a *.an* extension, where *n* indicates the numerical order in which you defined the area. After it uses the last area, the database engine reuses the first area if it is empty. Figure 11–1 illustrates this behavior. An *extent switch* is the operation of switching from one AI area extent to another.

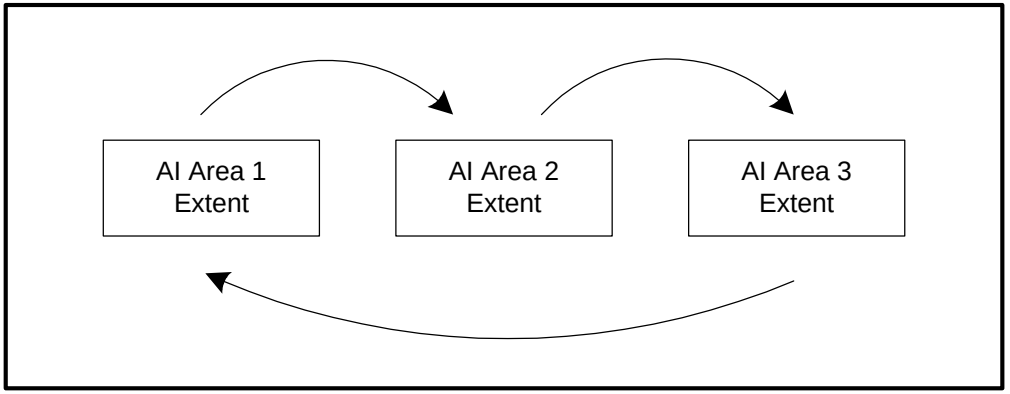


Figure 11–1: After-image Extents Switching

You must monitor the status of the extents to ensure that you do not try to reuse an unavailable file. For information on monitoring the status of your AI extents, see the "Monitoring AI File Status" section in this chapter.

Like DB and BI extents, there are two types of AI extents:

- Fixed length
- Variable length

Fixed-length Extents

Fixed-length extents are extents that you preallocate and preformat in the structure description file. With fixed-length extents, you control how much disk space each extent uses.

Variable-length Extents

Variable-length AI extents do not have a predefined length. They continue to fill until they use the entire disk, you back up the database, or you issue the RFUTIL AIMAGE NEW command. The initial length of a variable-length extent is 128K. Like DB areas, you can define more than one variable-length AI area for a single database.

11.2 Estimating After-imaging Disk Space Requirements

Before creating after-image (AI) files, it is important to accurately estimate the amount of disk space required to hold all the AI data for your database. Even when you use variable extents, if the disk runs out of space and there is no empty AI extent available, you must perform emergency maintenance or the database is forced to shut down. See Chapter 8, "Recovering a Database," for details. To prevent the database engine from shutting down when it exhausts AI disk space, start your database with the after-image stall (`-aistall`) startup parameter.

To determine the amount of AI file space required, use the PROMON utility BI Log Activity Display to monitor the number of BI writes that occur during the activity period that you want to use after-imaging. The information that is written in the BI file closely matches what is written in the AI file. Typically 0.05% fewer bytes are written to an AI file than are written to the BI file. Therefore, the BI Bytes written statistic provides a close estimate of the space required for an AI file for the same activity period.

Measure the BI Bytes written statistic several times toward the end of the activity period that you are considering using after-imaging. If your workload varies from day to day or peaks during end-of-month processing, count the BI writes during a peak period to ensure that you allocate enough space to handle peak days. If the workload grows, make sure you increase the amount of space available for AI files. Always have extra disk space available for unanticipated growth of the AI file.

11.3 Creating After-image Areas

Follow these steps to create AI areas:

1 ♦ Create the structure description file for the database. See Chapter 4, "Creating and Deleting Databases," for a complete description of how to create a structure description file.

2 ♦ Define the data extents.

3 ♦ Define any BI extents.

4 ♦ Define any fixed-length AI extents. You must define four tokens in the file description line. Table 11–1 describes each token and what you have to enter for fixed-length files.

Table 11–1: File Tokens

Token	Description
First	Specifies the file type. Type **a** to indicate that this extent is an AI extent.
Second	Specifies the extent file pathname. Enter a pathname that represents a standard operating system file. If you do not supply an extension, the database engine automatically appends a .a*n* extension.
Third	Specifies whether the extent is a fixed-length or variable-length extent. Type **f** for a fixed-length extent. It must be lowercase.
Fourth	Specifies the length of the extent in 1,024-byte (1K) units. The minimum length for a fixed-length extent is 16K. The extent length must be a multiple of 16K for all operating systems. If you specify any other size, PROSTRCT displays a warning message and rounds the size up to the next multiple of 16K. The maximum length of an extent depends on the size of the file system and/or physical volume that contains the extent.

The following example shows a fixed-length after-image extent definition:

```
a    db/mary/apdir/test.a1 f 2048
```

5 ♦ Define any variable-length after-image extents. You only have to specify the first two tokens: the extent type and pathname. Unlike data or BI extents, you can define more than one variable-length after-image extent for a single database.

6 ♦ Create the empty database structure using the PROSTRCT utility. For more information about the structure description file and the PROSTRCT and PROCOPY utilities, see Chapter 9, "Maintaining Database Structure."

7 ♦ Create the initial database contents using the PROCOPY or PROREST utility. For more information about the PROCOPY and PROREST utilities, see Chapter 19, "Database Administration Utilities."

11.4 Enabling After-imaging

Use the RFUTIL utility with the AIMAGE BEGIN qualifier to enable after-imaging. The use of after-imaging requires a recent backup of the database because the database backup and AI files are treated as a recoverable set of files.

Follow these steps to enable after imaging for a database:

1 ♦ Define AI areas in the structure description file. See the "Creating After-image Areas" section for detailed instructions.

2 ♦ Back up the database.

3 ♦ After you create the database, use the following syntax to enable after-imaging:

```
rfutil db-name -C aimage begin
```

db-name

Specifies the name of the database you are using. If the database has been modified since it was last backed up, RFUTIL displays an error message and does not enable after-imaging.

11.5 Managing After-imaging Files

The primary tasks you must perform to manage after-imaging are:

- Monitor the AI extent status.
- Switch to a new extent.
- Archive the AI extents.
- Make a full AI extent available for reuse.

The following sections detail each task.

11.5.1 Monitoring AI File Status

You must monitor the status of the AI extents. The AI extent status can be one of the following:

- **Empty** — When an AI extent is empty, it is available for use.
- **Busy** — When an AI extent is busy, it is currently in use.
- **Full** — When a busy AI extent fills up, an extent switch occurs: the state of the AI extent changes to full and the next AI extent becomes busy.

Monitoring AI Extent Status

Use the RFUTIL AIMAGE EXTENT LIST utility to display information about each extent, including status. This command returns no information if after-imaging is disabled:

```
rfutil db-name -C aimage extent list
```

For more information, see the description of the AIMAGE EXTENT LIST utility in Chapter 19, "Database Administration Utilities."

Use the RFUTIL AIMAGE EXTENT FULL utility to provide the filename of the oldest full file. You can then use this information to archive extents in the order in which they were filled. Although there might be multiple full files, this command displays the pathname of the oldest full file:

```
rfutil db-name -C aimage extent full
```

For more information, see the description of the AIMAGE EXTENT FULL utility in Chapter 19, "Database Administration Utilities."

11.5.2 Switching To a New AI File

You switch to a new AI extent for the following reasons:

- As part of the backup schedule
- When the current fixed-length AI extent is full, or when the disk holding the current variable-length AI extent is full
- Before archiving an AI extent

Except when you switch to a new extent because the current extent is full, switching to a new AI extent establishes a starting point for backup; after you restore the backup, you roll forward starting from that extent.

NOTE: When you perform an online backup, PROBKUP automatically switches over to a new extent as long as the next extent is empty. Before you perform the online backup, make sure that the next extent is empty.

A fixed-length extent has a predefined size, so the database engine can determine when the extent becomes full.

In contrast to a fixed-length extent, a variable-length extent does not have a predefined maximum size. Therefore, the database engine cannot anticipate when the extent is about to become full. Unless you force a switch using the RFUTIL AIMAGE NEW utility, the database engine continues writing to the extent until there is no more room left on the disk or the 2GB addressable AI file limit is reached. When the extent becomes full, the database engine automatically switches to the next extent, provided that the next extent is empty.

If the next extent is full, the database engine shuts down the database. However, you can use the After-image Stall (`-aistall`) parameter to suspend database activity and send a message to the log file or you can use the RFUTIL qualifier AIMAGE AIOFF to disable after-imaging. If you use `-aistall`, you can archive the extent and mark it as empty. The system will then automatically switch to that extent and the database activity automatically resumes. For more information on the `-aistall` parameter, see Chapter 18, “Database Startup Parameters.” If you use RFUTIL AIMAGE AIOFF, after-imaging becomes disabled and can no longer write notes.

NOTE: You can only use the `-aistall` parameter and RFUTIL AIMAGE AIOFF in multi-user mode.]

When the database engine suspends database activity or shuts down the database, it sends the following message to the log file:

```
Can't switch to after-image extent filename it is full.
Backup ai extent and mark it as empty (3774)
```

The database engine cannot resume database activity until the next extent is backed up and marked as empty.

Manual Switching To a New AI Extent

You can manually perform an online AI extent switch if you want to archive the AI file at regularly scheduled times instead of waiting until the extent becomes full.

Follow these steps to switch to the next extent in the sequence:

1 ♦ Make sure the next extent in the sequence is archived. If not, archive it. See the "Archiving an AI File" section for details.

2 ♦ Use the RFUTIL AIMAGE NEW utility with the following syntax:

```
rfutil db-name -C aimage new
```

When you issue the RFUTIL AIMAGE NEW command, RFUTIL changes the status of the current extent to full and changes the status of the next file to busy. For more information on this command, see Chapter 19, "Database Administration Utilities."

11.5.3 Archiving an AI File

Backing up the AI file involves:

- Scheduling the backup
- Performing the backup
- Marking a file as empty
- Labeling the backup

Scheduling Backups

Depending on your database backup schedule and needs, you might want to back up the AI files:

- Each time you perform a database backup.

  Back up the AI files each time you back up the database. It should be a routine part of every backup. Also, back up the AI files on a backup media different from the database, log, and BI files. This technique provides a way to reconstruct the database by rolling forward the backup of the AI files against the previous backup of the database.

  If you do choose to back up the AI files on the same media as the database, it is essential that you back up the AI files in a way that lets you restore it separately from the database, BI, and log files.

  NOTE: If you back up the AI files when you perform a database backup, keep in mind that the AI files backup is used with the previous database backup.

  You can use the RFUTIL utility commands with online databases that have AI extents. You can perform online backups while after-imaging is enabled. The backup utility automatically marks the busy AI file as full and switches over to a new AI file.

- On a daily basis.

  You should consider backing up the AI files on a daily basis if:

  – You are on a weekly full backup schedule and the AI files will grow very large during the week.

  – You want to perform daily backups of the AI files instead of performing incremental backups. However, it is quicker to restore your database from incremental backups than AI backups.

Before deciding to back up the AI files every day, consider that recovering the database from these small AI files is more intricate than recovering from a single, large AI file. Also, if you are using a single AI file, you must shut down the database to switch to a new AI file.

If you are using multiple AI extents, you must back up the extents regularly to ensure that the system does not run out of AI space. If you are using a single AI file, it is important to back up the AI file before you fill the disk that contains it. If you do not back it up in time, the database engine shuts down the database. For a complete description of how to recover from a full AI disk, see Chapter 8, “Recovering a Database.”

Performing the Backup

You must use an operating system utility to back up the AI files regardless of whether you are using a single AI file or multiple AI files. Table 11–2 presents some of the backup options available for each operating system.

Table 11–2: Operating System Backup Utilities

Operating System	Available Backup Utilities
UNIX	`tar`, `cpio`, or a manufacturer-supplied backup utility.
Windows	Windows Backup or any backup utility that can back up and restore individual files.

Ensure that the backup technique backs up the entire file. On many UNIX systems, certain utilities (for example, `cpio`) will back up only the first part of files over a certain size (controlled by the ULIMIT parameter). Backups of partial AI files are invalid and unusable.

Marking an AI File As Empty

After you back up a file, you must mark the file as empty so that the database engine can reuse the file. To mark a file as empty, use the AIMAGE EXTENT EMPTY qualifier of the RFUTIL utility. After you mark the file as empty, the database engine overwrites the contents of the file.

Use the following command to mark the AI file as empty:

```
rfutil db-name -C aimage extent empty [ extent-number | extent-path ]
```

db-name

Specifies the database you are using.

extent-number

Specifies the number of the file you want to mark as empty.

extent-path

Specifies the pathname of the file you want to mark as empty.

If you do not specify an *extent-number* or *extent-path*, the database engine automatically marks the oldest full extent as empty.

NOTE: Use the RFUTIL AIMAGE EXTENT LIST or RFUTIL AIMAGE EXTENT FULL utility to determine the *extent-number* or *extent-path*.

Labeling the Backup

After you back up the AI file, you must properly label the backup. Properly labeling backups helps you ensure database integrity. Follow these steps to label backups:

1 ♦ Clearly label each backup. Information on the label might include the:

- Date and time of the backup
- Name of the database
- Name and type of the file
- Volume number and total number of volumes of the media, even if there is only one volume
- Initials of the person who made the backup
- Utility to use to restore the backup

2 ♦ Keep all daily database and AI file backups for at least two weeks. In the event that you have to roll the database forward, you can then reapply each of these daily backups to recover the lost data. If you do not keep a daily backup of the AI files and you lose the database and AI files, the most recent backup might be as much as a week old. However, if you have been doing daily backups of the AI files, you can use the most recent copy of that file to reconstruct a more recent version of the database.

3 ♦ Keep backups in an area other than where the computer is located, preferably in another building. Thus, in the event of building damage, you are less likely to lose both the online and backup versions of the files.

11.6 Performing Roll-forward Recovery

To perform roll-forward recovery, roll forward all of the extents used since the last backup, in the order they were filled. You must roll forward all of the extents before you can start the database. If you inadvertently start a database before you restore all of the AI extents, the database engine rolls back any incomplete transactions. Although these transactions might be completed in other as yet unrestored AI files, they appear to the database engine as incomplete during the roll-forward process. If you start the database before you restore all of the extents, you must begin the roll forward again from the last backup.

Before performing roll-forward recovery, you must have the most recent database backup, all AI files generated since the most recent backup, and no missing files. The last file is usually the file in use when the system failed. If any AI file is missing, recovery can be performed up to the missing file, but no further.

Use the ROLL FORWARD qualifier of the RFUTIL utility to restore each file:

```
rfutil db-name -C roll forward -a ai-name
```

db-name

Specifies the database you are using.

ai-name

Identifies the AI file of the specified database.

The ROLL FORWARD qualifier fails if:

- You omit the After-image Filename (-a) parameter
- It cannot open the AI file
- You name the wrong AI file

If the system fails while you are running the ROLL FORWARD operation, restore the database files again and rerun the ROLL FORWARD operation.

The ROLL FORWARD qualifier always disables after-imaging for the database before beginning the roll-forward operation. After the roll-forward has completed, you must re-enable after-imaging with the AIMAGE BEGIN qualifier if you want continued AI protection.

To perform a partial roll-forward recovery, use the endtime or endtrans options. The endtime option lets you roll forward an AI file to a particular point. The endtrans option lets you roll forward an AI file to a particular transaction.

For more information about the ROLL FORWARD qualifier, see Chapter 8, "Recovering a Database" and the entry for the RFUTIL utility in Chapter 19, "Database Administration Utilities."

11.7 Disabling After-imaging

Follow these steps to disable after-imaging:

1 ♦ If you have a database, back up any full extents. This step is important because when you disable after-imaging on a database with AI extents, the database engine marks all of the AI extents as empty. This means that you cannot access any information that you did not back up for roll-forward recovery.

2 ♦ If you have a database, back up the extent that is currently busy.

3 ♦ Disable after-imaging. Use the AIMAGE END qualifier with the RFUTIL utility:

```
rfutil db-name -C aimage end
```

db-name

Specifies the database you are using.

You can also use the AIMAGE AIOFF qualifier with RFUTIL. Use AIMAGE AIOFF when you need to temporarily disable after-imaging, such as during scheduled maintenance:

```
rfutil db-name -C aimage aioff
```

db-name

Specifies the database you are using.

12

Using Two-phase Commit

The Progress database engine uses two-phase commit to ensure database integrity when updating two or more databases in a single transaction. Two-phase commit is not necessary for transactions involving a single database.

This chapter contains the following sections:

- Distributed Transactions
- How The Database Engine Implements Two-phase Commit
- Enabling Two-phase Commit
- Detecting Limbo Transactions
- Resolving Limbo Transactions
- Deactivating Two-phase Commit
- Case Study

12.1 Distributed Transactions

Two-phase commit ensures that distributed transactions occur consistently across all databases. A *distributed transaction* is a single transaction that updates two or more databases. The following 4GL procedure is an example of a distributed transaction:

disttran.p

```
DO TRANSACTION:
   FIND FIRST sports1.customer.
   sports1.customer.credit-limit = sports1.customer.credit-limit - 10.
   FIND sports2.customer WHERE sports2.customer.cust-num =
       sports1.customer.cust-num.
   sports2.customer.credit-limit = sports2.customer.credit-limit + 10.
END.
```

The following scenario illustrates how inconsistencies can occur during a distributed transaction. A bank has two accounts, one on database sports1 and another on database sports2. The bank runs an application that starts a transaction to withdraw a sum of money from sports1 and deposit it into sports2. To keep the accounts in balance, it is critical that both operations—the withdrawal and the deposit—succeed, or that they both fail. For example, if sports1 commits its part of the transaction and sports2 does not, there is an inconsistency in the data, as shown in Figure 12–1.

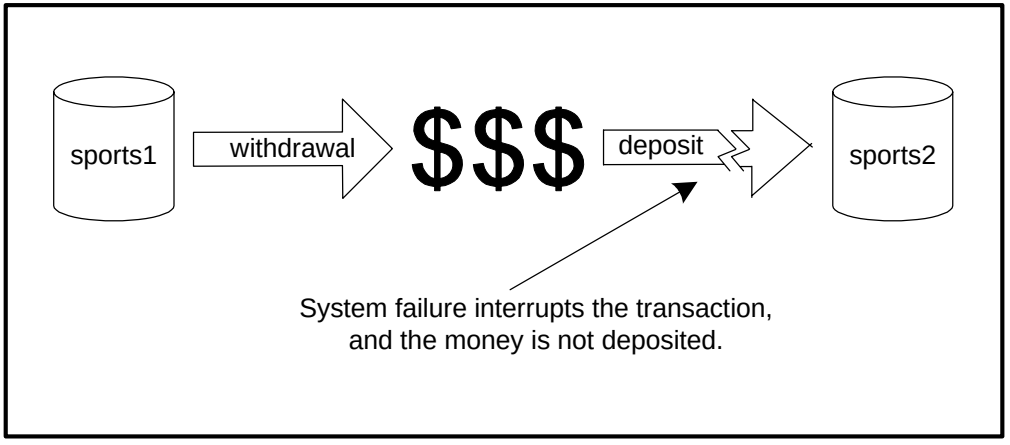


Figure 12–1: Data Inconsistency

Two-phase commit protects against this type of inconsistency by making sure that all databases commit the transaction, or that none commit. To ensure database integrity across all involved databases, the database engine commits database updates in two distinct phases. During the first phase, the database engine checks each database involved in a transaction to verify that it is ready to commit the transaction. During the second phase, the database engine directs the databases to commit the transaction and then verifies that they committed it properly.

If there is an inconsistency, the database engine displays error messages and allows you to complete or roll back the inconsistent transaction to return the data to a consistent state.

12.2 How The Database Engine Implements Two-phase Commit

To implement two-phase commit, the database engine assigns a coordinator database and a transaction number for each distributed transaction. The *coordinator database* establishes the transaction's status (either committed or terminated). The *transaction number* identifies individual transactions. The transaction number for a distributed transaction is the transaction number assigned to the coordinator database. The database engine stores the name of the coordinator database and the transaction number in the before-image (BI) file of each database involved in the distributed transaction.

The coordinator database establishes the status of the transaction by committing or terminating the transaction. The action that the coordinator database takes is final and irreversible. If the coordinator database commits a transaction, the status of the transaction is committed, even if other databases do not commit the transaction. (This can happen because of a hardware or software failure.) Likewise, if the coordinator terminates a transaction, the status of the transaction is aborted. All of the other databases involved in the transaction must perform the same action as the coordinator database, or your data will be inconsistent.

Figure 12–2 shows the algorithm that the database engine uses to implement two-phase commit. Because this algorithm requires additional unbuffered I/O operations to ensure transaction integrity, there might be a performance impact when you implement two-phase commit. If two-phase commit is enabled, transactions that do not involve multiple databases do not incur additional I/O for those transactions.

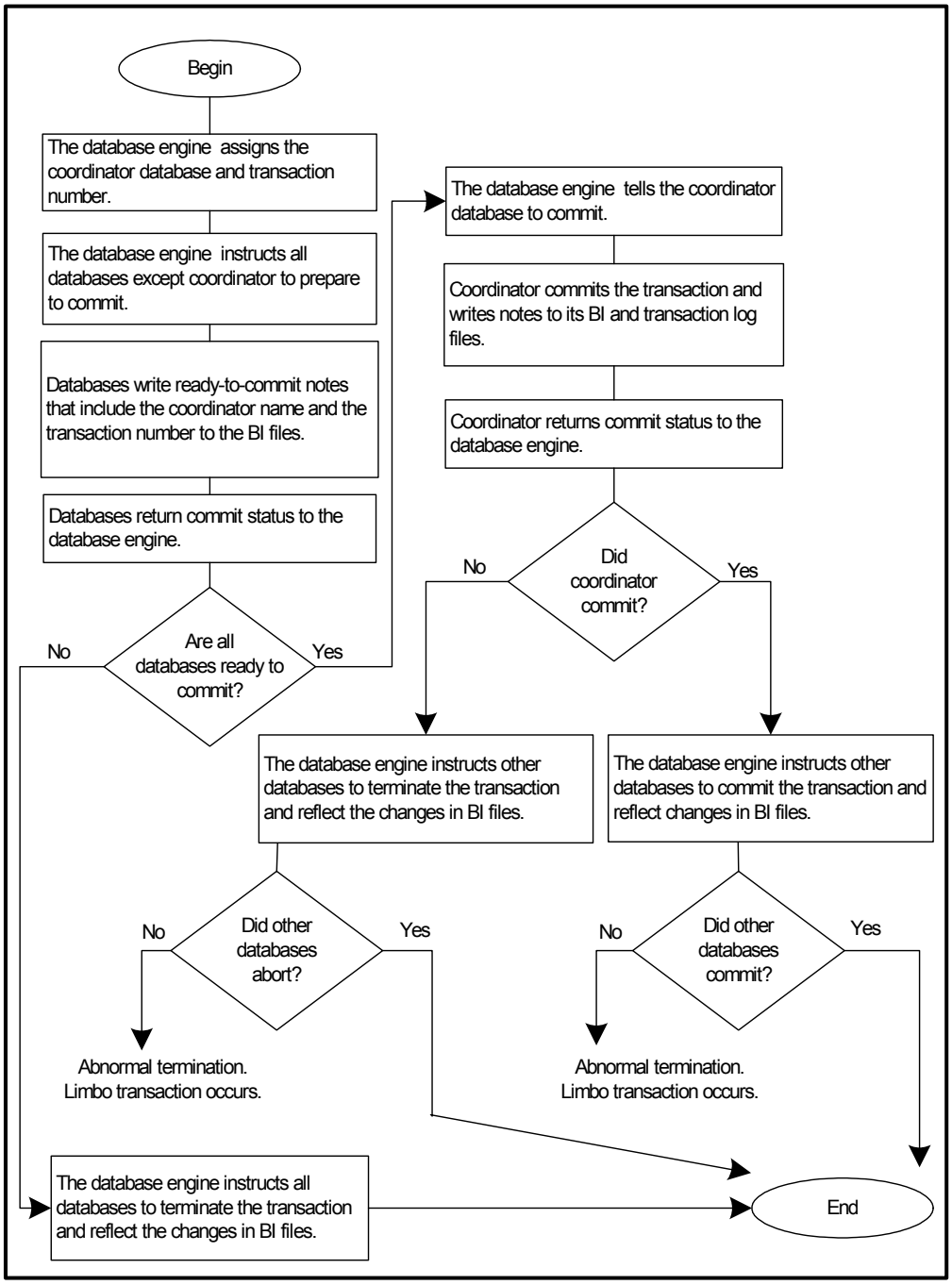


Figure 12–2: Two-phase Commit Algorithm

As Figure 12–2 shows, the coordinator database is the first database in the distributed transaction to either commit or abort the transaction. By keeping track of the coordinator's action, two-phase commit allows you to resolve any inconsistent transactions.

A *limbo transaction* (also known as an in-doubt transaction) occurs if the coordinator database commits or aborts a distributed transaction, but a hardware or software failure prevents other databases from doing likewise. This is called a limbo transaction because the processing of the transaction is temporarily suspended. A limbo transaction might occur for a variety of reasons, for example, as a result of a power outage. Figure 12–3 illustrates a limbo transaction.

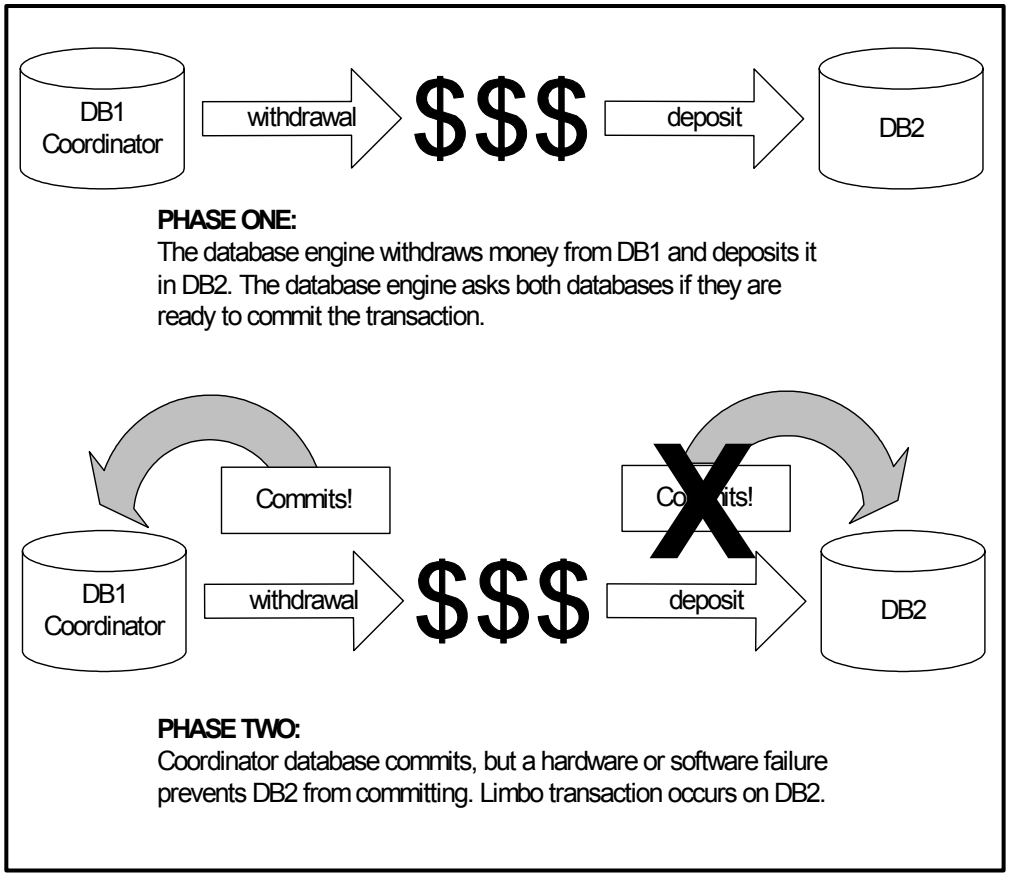


Figure 12–3: Limbo Transaction

When a limbo transaction occurs, you must resolve the transaction to re-establish data consistency. You resolve a limbo transaction with the Progress Database Monitor (PROMON) utility or with the database engine.

Once the coordinator database establishes the status of a distributed transaction, it writes the status to its BI and TL (transaction log) files. The TL file tracks the status of all distributed transactions that affect the coordinator database.

Since the database engine continually overwrites the contents of the BI file, the TL file is necessary to permanently record the status of a transaction. If you must resolve limbo transactions, the transaction log file ensures that the coordinator has a reliable record of the transaction.

If you enable after-imaging for the coordinator database, the coordinator automatically uses the after-image (AI) file to log the status of each distributed transaction, instead of using the transaction log file. By using the AI file, the database engine writes to disk less often than if you use both the AI and transaction log file, thus improving performance. However, the database engine still uses the transaction log file to store information when you make an AI file available for reuse. In addition, if you disable after-imaging, the coordinator once again uses the transaction log file.

12.2.1 Two-phase Commit and Roll-forward Recovery

If you use two-phase commit, you should also use after-imaging to ensure database integrity and avoid backup synchronization problems. Although two-phase commit ensures that distributed databases remain synchronized, it does not protect you from a lost database or BI file. If you lose an entire file or disk, you must use the AI file and roll-forward recovery to return to the point of the crash.

Keep the following information in mind when performing roll-forward recovery using RFUTIL:

- If you perform roll-forward recovery on a database that has after-imaging and two-phase commit enabled, RFUTIL disables after-imaging and two-phase commit.
- When you roll forward an after-image file that contains coordinator transaction end notes, RFUTIL writes a transaction log file containing the notes. Also, if two-phase commit is not enabled, RFUTIL enables two-phase commit for the coordinator database.

See Chapter 8, “Recovering a Database,” for more information about roll-forward recovery and after-imaging.

12.3 Enabling Two-phase Commit

PROUTIL provides two-phase commit protection only if you enable two-phase commit on two or more of the databases involved in a distributed transaction. For example, if a transaction involves three databases and you enable two-phase commit for two of them, PROUTIL provides two-phase commit protection for the two databases. However, PROUTIL protects only the databases that you enable, so the transaction is not completely protected from failure. For true integrity, enable two-phase commit for all three databases.

NOTE: You must create and maintain a transaction log (TL) area for your database in order to use two-phase commit. For more information, see the "Transaction Log Area" section. For a complete description of database storage areas see Chapter 1, "The Progress Database."

You enable two-phase commit with the PROUTIL 2PHASE BEGIN qualifier. When you enable two-phase commit, you can specify the database that should serve as the coordinator database. You can also specify an alternate name (nickname) for the coordinator database.

This is the syntax for enabling two-phase commit:

```
proutil db-name -C 2phase begin [ -crd | -tp nickname ]
```

db-name

Specifies the database you are using.

-crd

Specifies that the database can serve as a coordinator database. For example, if you enable two-phase commit for three databases (db1, db2, and db3) and you specify the -crd parameter for db3, PROUTIL assigns db3 as the coordinator database. However, if you specify the -crd parameter for more than one database, PROUTIL arbitrarily assigns a coordinator database from the databases that received the -crd parameter. If you do not assign any database as a coordinator, all two-phase-commit-enabled databases are potential coordinator databases. PROUTIL randomly assigns a coordinator database from one of these databases.

Choose a database on a reliable machine to serve as the coordinator database. This way, you can be more certain that you have a reliable record of all transactions that occur. You can also be more certain that a hardware or software failure does not affect the coordinator database.

-tp

Specifies a unique nickname that PROUTIL uses to identify the coordinator database. If you do not specify a nickname, PROUTIL automatically chooses the name of the database (without the .db extension) as the nickname.

For example, if you have a database named /usr/dbs/appl.db, the nickname for the database is appl. If PROUTIL assigns appl.db as the coordinator database, it writes the nickname appl to the BI file instead of the database's full path name. Specify nicknames of up to eight characters. Specifying a shorter nickname decreases the size of the notes that must be written.

NOTE: Be sure to specify a unique nickname. If you must resolve limbo transactions with two databases that have the same path name but are on different machines, PROUTIL does not distinguish between the two databases.

12.3.1 Modifying the Database Nickname and Priority

If you want to change the nickname of a database, or if you want to change the priority of your databases, use the 2PHASE MODIFY qualifier of the PROUTIL utility:

```
proutil db-name -C 2phase modify [ -crd | -tp nickname ]
```

db-name

Specifies the database you are using.

-crd

Switches whether or not the database can serve as a coordinator database. If you specify -crd against a database that is a candidate for coordinator database, it is no longer a candidate. If you specify -crd against a database that is not a candidate, it becomes a candidate.

-tp

Identifies a new nickname for the coordinator database.

12.3.2 Transaction Log Area

A separate transaction log (TL) storage area holds the transaction log data generated when two-phase commit is in use. You must create and maintain a TL area for your database in order to use two-phase commit.

The transaction log contains the transaction number of all committed distributed transactions. When there is an in-doubt transaction, this log is scanned for the transaction number to determine if the transaction committed or not. If the transaction number is found in the log, it means the transaction committed. If the transaction number is not found in the log, it means the transaction aborted.

When the last block of the TL area is written, PROUTIL automatically resets the current write position to the first block of the area. Thus, the oldest transaction recorded in the log is in the block after the current write position once the write pointer has wrapped around for the first time. The record of the transaction commit written to the log is the 32-bit transaction id of the committed transaction. Aborted transactions do not have a record written to the log.

The TL area can be composed of one or more fixed-length extents. The TL area cannot have a variable-length extent. The TL block size is 16K. Each 16K block can contain the commit record for 4,000 distributed transactions. The transaction log should contain enough space so that the record of a committed transaction is not overwritten before a database containing an in-doubt transaction can be brought back online.

To determine how large the TL area should be, use the PROMON utility to determine the average number of transactions in a 24-hour period. Multiply that number by 1.2, then round up to the nearest multiple of 16KB. The result is the number of bytes you should specify for your TL extents.

For example, if a site commits 1,000,000 transactions per day, the TL extent size needed is:

1,000,000 * 1.2 = 1,200,000
1,200,000/16384 = 73.242 (or 74 rounded up)
74 * 16384 = 1212416
1212416/1024 = 1184 KB

This indicates that the total extent length of the TL area should be set to 1184K.

12.4 Detecting Limbo Transactions

When a limbo transaction occurs, PROUTIL writes information about the failed transactions to the log (.lg) file and displays one of the following messages:

```
Part of the distributed transaction might have failed.(2022)
Cannot roll back a limbo transaction.(2030)
```

Figure 12–4 illustrates where messages are displayed.

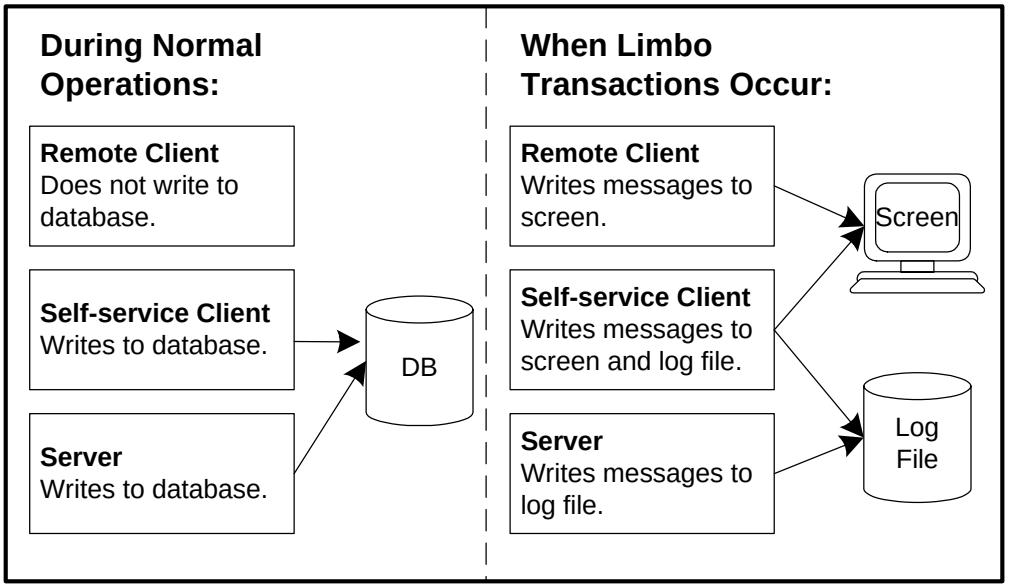


Figure 12–4: How PROUTIL Processes React To Limbo Transactions

Limbo transactions can occur without any messages being displayed on screen, for example, if a hardware or software failure occurs while a user is running a PROUTIL application or if a user powers off a client machine. If possible, users on client machines should inform the system administrator when these events occur. If such an event occurs, examine all of the databases that might be involved to determine whether any limbo transactions occurred. You can use PROMON or PROUTIL to examine a database for limbo transactions.

CAUTION: If an application is performing a distributed transaction when a client machine fails or shuts down, the transaction remains open. If this continues unchecked, the BI files of the databases involved in the transaction could grow to fill the disk, as with any other long-running transaction.

12.5 Resolving Limbo Transactions

When you detect a limbo transaction, follow these steps to resolve it:

1 ♦ Determine the transaction numbers and the coordinator database.

2 ♦ Find out if a limbo transaction was committed in the coordinator database.

3 ♦ Resolve the limbo transaction by committing or aborting it in the limbo database.

There are two techniques for resolving limbo transactions. You can:

- Resolve a limbo transaction on a database that is still in use with PROMON.
- Resolve limbo transactions on a database that is shut down. For this technique, try to start a session against the database, then use PROUTIL 2PHASE RECOVER.

12.5.1 Resolving Limbo Transactions With the Database In Use

Use PROMON to resolve limbo transactions while the database is still in use. You can only run PROMON against a database if a server is running against the database. If no server is running, or if at any point during this procedure PROMON will not run against a database, see the "Resolving Limbo Transactions With the Database Shut Down" section.

Follow these steps to resolve limbo transactions with PROMON:

1 ♦ Determine whether one or more limbo transactions occurred against a database by starting the PROMON database monitor. Enter the following command:

```
promon db-name
```

db-name

Specifies the database you want to monitor.

When you enter the PROMON utility, the main menu appears:

```
PROUTIL MONITOR Version 9
Database: /usr/dlc/sports

1.  User Control
2.  Locking and Waiting Statistics
3.  Block Access
4.  Record Locking Table
5.  Activity
6.  Shared Resources
7.  Database Status
8.  Shut Down Database

T.  Transaction Control
L.  Resolve Limbo Transaction
C.  Coordinator Information

M.  Modify Defaults
Q.  Quit

Enter your selection:
```

2 ♦ Choose option T (Transaction Control). PROUTIL displays a screen similar to this:

```
1.  Display all entries
2.  Match a user number
3.  Match a range of user numbers
Q.  Return to main menu

Enter your selection:
```

3 ♦ Choose option 1 (Display all entries). PROUTIL displays a screen similar to this:

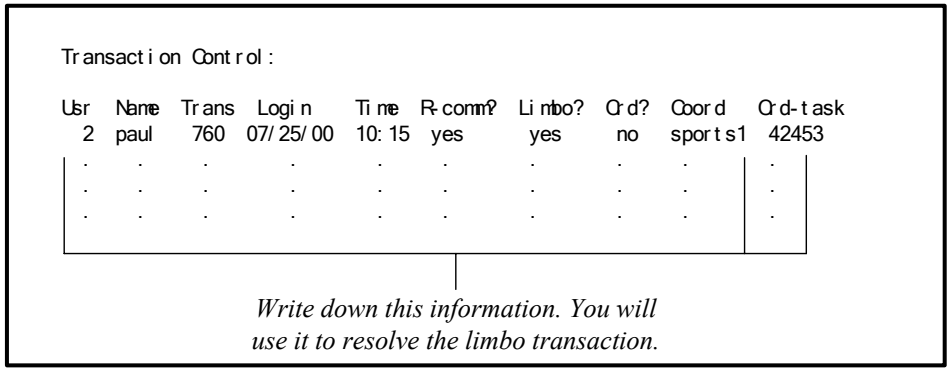
```
Transaction Control:

Usr  Name  Trans  Login     Time   R-comm?  Limbo?  Crd?  Coord    Crd-task
  2  paul    760  07/25/00  10:15  yes      yes     no    sports1  42453
  .    .      .         .      .      .        .       .      .        .
  .    .      .         .      .      .        .       .      .        .
  .    .      .         .      .      .        .       .      .        .
```

A limbo transaction displays yes in this field.

NOTE: If you run PROMON against a database where no limbo transaction has occurred, PROMON does not display any field information on the Transaction Control screen.

Take note of any limbo transactions. For example:



A transaction is in limbo if yes is displayed in the Limbo field. For each limbo transaction, write down the following information:

- The user number, shown in the Usr field
- The name of the coordinator database, shown in the Coord field
- The transaction number of the transaction in the coordinator database, shown in the Crd-task field

You need this information to resolve the limbo transaction.

To resolve limbo transactions, you must consult the coordinator database of each transaction to see if the coordinator committed the transaction. If the coordinator database committed the transaction, you must also commit the transaction on the database where the limbo transaction occurred. If the coordinator did not commit the transaction, you must terminate the transaction on the database where the limbo transaction occurred.

4 ♦ For each limbo transaction, run PROMON against the coordinator database to determine whether the coordinator committed the transaction.

5 ♦ From the PROMON main menu, choose option C (Coordinator Information). PROUTIL displays a screen similar to this:

```
PROUTIL MONITOR Version 9

     Database: /users/sports1

     Q. Quit

Enter the transaction number you want to find out if committed:
```

NOTE: If the coordinator database is shut down and you cannot run PROMON against it, you must use the 2PHASE COMMIT qualifier of PROUTIL to determine whether it committed the transaction. For more information, see the "Resolving Limbo Transactions With the Database Shut Down" section.

6 ♦ Enter the transaction number that you recorded from the Crd-task field in Step 3, and press **RETURN**. PROUTIL displays a message that tells you whether the transaction committed.

NOTE: To commit transactions on a database that is shut down, you must use the 2PHASE RECOVER qualifier of PROUTIL. For more information, see the "Resolving Limbo Transactions With the Database Shut Down" section.

7 ♦ Run PROMON against the database where the limbo transaction occurred to commit or abort each limbo transaction.

8 ♦ From the PROMON main menu, choose option L (Resolve Limbo Transactions). This menu appears:

```
                  1  Abort a Limbo Transaction
                  2  Commit a Limbo Transaction
                  Q  Quit

            Enter choice>
```

9 ♦ To commit the transaction, choose 2 (Commit a Limbo Transaction). PROUTIL prompts you to enter the user number you recorded in Step 3, then press **RETURN**. PROUTIL displays a message similar to this:

```
User 1: commit transaction and disconnect.
```

To abort the transaction, choose 1 (Abort a Limbo Transaction). PROUTIL prompts you to enter the user number of the transaction you want to abort. Enter the user number, then press **RETURN**.

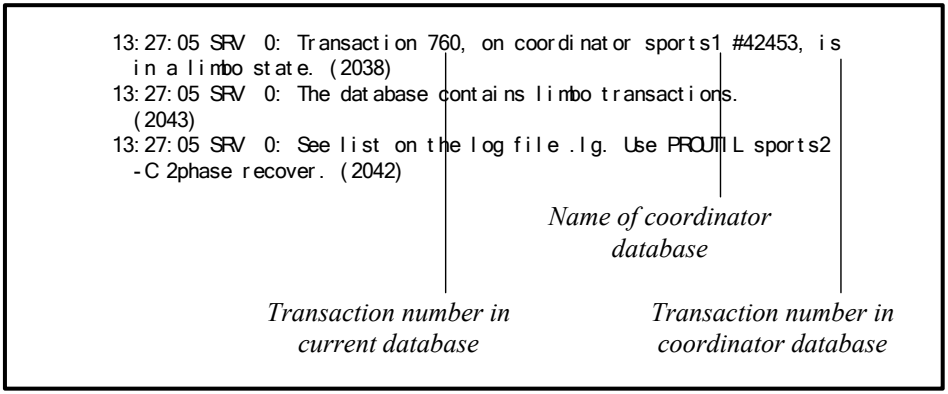
Repeat Steps 4 through 9 for all the limbo transactions. After you commit or abort all of the limbo transactions, they are resolved.

12.5.2 Resolving Limbo Transactions With the Database Shut Down

To resolve limbo transactions for a database that is shut down, use the PROUTIL utility. If a server is running against the database, use PROMON to resolve the transactions.

Follow these steps:

1 ♦ To determine whether one or more limbo transactions occurred against a database that is shut down, try to start a PROUTIL session against the database. If limbo transactions occurred, the PROUTIL session fails to start and PROUTIL displays output similar to the following, or writes it to the event log file for the database. If the PROUTIL session starts successfully against a database, no limbo transactions have occurred on the database:



For all the listed limbo transactions, capture the following information:

- The transaction number on the current database (that is, the database where you tried to start the PROUTIL session)
- The name of the coordinator database
- The transaction number in the coordinator database

Once you have this information, you must consult the coordinator database to determine whether it committed or aborted the transaction.

2 ♦ Enter the following command against the coordinator database to determine if the coordinator committed or aborted the limbo transaction:

```
proutil db-name -C 2phase commit tr-number
```

db-name

Specifies the coordinator database.

tr-number

Specifies the number of the transaction to check. Specify the number of the transaction on the coordinator database.

If the coordinator committed the transaction, PROUTIL displays a message similar to this:

```
Transaction 42453 has committed. (2048)
```

If the coordinator database committed the transaction, you must also commit the transaction on the database where the limbo transaction occurred. If the coordinator did not commit the transaction, you must abort the transaction on the database where the limbo transaction occurred.

3 ♦ Commit or abort the limbo transactions, depending on whether the coordinator committed or aborted the transaction in Step 2.

Use the PROUTIL 2PHASE RECOVER utility to commit or abort transactions for a database. Before you enter this command, determine whether you will commit or abort each transaction; you must either commit or abort all limbo transactions to complete this command:

```
proutil db-name -C 2phase recover
```

db-name

Specifies the database with the limbo transaction.

When you run this command against a database with limbo transactions, PROUTIL displays a message similar to this:

```
Commit transaction 760, on coordinator sports1 #42453 (y to commit/n to
abort)? (2039)
```

If you respond yes, PROUTIL commits the transaction. If you respond no, PROUTIL aborts the transaction.

PROUTIL displays this message for all of the limbo transactions that exist in the database.

After you commit or abort all of the limbo transactions, they are resolved.

12.5.3 Resolving Limbo Transaction Scenarios

This section describes three scenarios in which you must resolve limbo transactions.

Scenario 1: You Are On a Client Machine and the Server Fails

If you are on a client machine running a distributed transaction and something goes wrong with the server, PROUTIL displays this message:

```
Part of the distributed transaction may have failed. (2022)
```

This message does not necessarily mean that a transaction failed. Occasionally, a transaction commits properly, but a network communication failure intercepts the server's message verifying that it committed. When you see this message, or any similar message, the database administrator must determine whether a limbo transaction occurred, then resolve the limbo transaction.

To resolve limbo transactions, you complete the transactions from the point where they were interrupted by the hardware or software failure. If the coordinator committed the transactions, you must commit the transactions. If the coordinator did not commit the transactions, you must abort the transactions.

Scenario 2: You Are Starting Up PROUTIL and Have a Power Failure

You have a power outage that shuts down all of the machines on your network. When the power is restored, you try to start a PROUTIL session. PROUTIL fails to open the database and displays a message similar to this:

```
13:27:04 SRV  0: Multi-user session begin. (333)
13:27:05 SRV  0: The database contains limbo transactions. (2043)
13:27:05 SRV  0: See list on the log file .lg. Use PROUTIL sports2 -C 2phase
  recover. (2042)
13:27:06 ** The server terminated with exit code 20. (800)
```

This message indicates that limbo transactions must be resolved. Consult the log file for a record of the limbo transactions.

Scenario 3: You Are On a Client Machine and It Fails

Suppose a hardware or software failure occurs on a running client machine, or a user inadvertently powers off a machine while the database is running. PROUTIL cannot display a message indicating that a limbo transaction occurred, since the client machine is down. In this situation, use the PROMON utility against the server to determine whether any limbo transactions occurred. If so, resolve them.

12.6 Deactivating Two-phase Commit

To deactivate two-phase commit for a database, use the 2PHASE END qualifier of the PROUTIL utility. The database cannot be in use:

```
proutil db-name -C 2phase end
```

db-name

Specifies the database where you want to disable two-phase commit. When you deactivate two-phase commit, PROUTIL places a note in the database log file. However, PROUTIL does not delete the database's transaction log file.

12.7 Case Study

This case study illustrates the process of resolving a limbo transaction. It involves two databases, sports1 and sports2, located on separate machines, mach1 and mach2, respectively. Each database has two-phase commit enabled; each has a server running against it. The coordinator database is sports1.

Suppose that you start a client process on mach1 against the sports1 database and then connect to the sports2 database using this command:

```
CONNECT sports2 -H mach2 -S sportssv
```

After connecting, you try to run a distributed transaction.

While running this procedure, the client process is halted by a system error, and the following messages appear:

```
Error reading socket, ret=-1, errno=2. (778)
Part of the distributed transaction might have failed. (2022)
Press space bar to continue.
```

The message indicates that a limbo transaction might have occurred. You must determine whether a limbo transaction did occur, then resolve it.

You start PROMON against sports1, choose T (Transaction Control), and choose 1 (Display all entries). This screen appears, indicating that there are no limbo transactions on sports1:

```
Transaction Control:
Usr Name  Trans  Login  Time  R-comm?  Limbo?  Crd?  Coord  Crd-task


RETURN - repeat, U - continue uninterrupted, Q - quit:
```

If PROMON failed to run against sports1, it indicates that the server also crashed and you must use PROUTIL to determine whether any limbo transactions occurred.

After determining that no limbo transactions occurred on sports1, perform the same steps against sports2. This time, the following screen appears, indicating that a limbo transaction has occurred:

```
Transaction Control:
Usr Name  Trans  Login  Time  R-comm?  Limbo?  Crd?  Coord  Crd-task
 15 paul  755  04/01/02 14:19  yes      yes     no  sports1  61061
                      .
                      .
                      .
RETURN - repeat, U - continue uninterrupted, Q - quit
```

Write down the coordinator's transaction number (indicated in the Crd-task field). The Coord field indicates that sports1 is the coordinator database for this transaction. Therefore, you must again run PROMON against sports1. This time, choose C (Coordinator Information). The following screen appears, where you enter the transaction number 61061:

```
PROGRESS MONITOR Version 9

   Database: /users/sports1

   Q. QUIT

Enter the transaction number you want to find out if committed: 61061
```

The following screen appears, indicating that the transaction committed:

```
Scan the logs...

** Transaction 61061 has committed.

Q. QUIT

Enter the transaction number you want to find out if committed:
```

Since the transaction committed on the coordinator sports1, you run PROMON against sports2 and choose 1 (Resolve Limbo Transactions). The following screen appears:

```
Transaction Control:
Usr   PID  Time of Login            User ID  TTY         Coord  Crd-task
 15  3308  Fri Apr 5 14:19:45 2002  paul   mach1 ttyp1  sports1   61061

     1 Abort a Limbo Transaction
     2 Commit a limbo Transaction
     Q Quit

Enter choice>
```

Choose 2 (Commit a Limbo Transaction), and this prompt appears:

```
Enter the user number whose transaction you want to commit:
```

Type **15** (the user number indicated on the previous screen). The PROMON utility commits the transaction on sports2 and displays this message:

```
User 15: commit transaction and disconnect.
```

Since there are no more limbo transactions, the situation is resolved and no further action is required.

13

Dumping and Loading

Dumping and reloading data definitions and table contents is important for both application development and database maintenance. This chapter details the different ways you can perform a dump and load.

Specifically, this chapter contains the following sections:

- Overview Of Dumping and Loading
- Dumping 4GL Database Definitions
- Dumping Database Contents
- Loading Database Definitions
- Loading Database Contents
- Bulk Loading
- Reconstructing Bad Load Records
- Specialized Dump and Load Techniques

NOTE: For a complete description of dumping and loading SQL-92 content, see the SQLDUMP and SQLLOAD descriptions in Chapter 19, “Database Administration Utilities.”

13.1 Overview Of Dumping and Loading

You might want to dump and load your database to:

- Create a new version of a database
- Use a constant table in multiple databases
- Economize disk space
- Migrate a database to a different version of Progress or a different operating system platform
- Load updated data definitions to upgrade a database schema

When you dump a database, you must first dump the database or table definitions, and then dump the table contents. The definitions and contents must be in separate files and cannot be dumped in one step. You perform both procedures with the Data Administration tool if you are using a graphical interface, or the Data Dictionary if you are using a character interface.

You can also dump and load the table contents with the PROUTIL command. This option dumps and loads the data in binary format.

4GL tools automatically attempt to disable triggers before performing the dump or load. If you do not have Can-Dump and Can-Load privileges, the 4GL tool asks if you want to dump or load the database without disabling the triggers. See Chapter 10, “Maintaining Security,” for information about assigning privileges.

13.1.1 Dump and Load Limitations

There are three restrictions for using the Dump and Load option from the 4GL tools:

- When you reload data from one database to another, the reload must be to a target database that has the **same** fields, arranged in the **same** logical order as the source database. The schema for each database must be identical. Otherwise, you must write your own reload procedure. You can use 4GL statements such as INPUT FROM, CREATE, and IMPORT.
- If you define a database field with a data type of ROWID or RECID, then the ROWID values in that field are dumped and reloaded as unknown value (?). You must write your own dump and reload procedures to accommodate ROWID fields.
- If you do not have Can-Create and Can-Write privileges for a file, you can still dump the data and data definitions for that file. You can also load the data definitions for the file, but you **cannot** load the data for that file.

NOTE: Progress can also dump and load table data in formats other than those described in this chapter. For information about other file formats Progress can dump and load, see the EXPORT and IMPORT statements in the *Progress Programming Handbook*.

13.2 Dumping 4GL Database Definitions

Use the Data Dictionary or Data Administration tool to dump database definitions. There are two ways to dump the database definitions:

- Dump the entire database, including all its tables, fields,
- Dump individual tables, sequences, or auto-connect records.

Whenever you run the dump utility to dump table definitions, the Data Dictionary or Data Administration tool creates a data definitions (DF) file that contains definitions of tables, fields, indexes, sequences, and auto-connect records, and all their characteristics. However, depending on whether you choose to dump all tables or only selected tables when you dump the definitions, the Data Dictionary or Data Administration Tool might or might not write all definitions to the DF file. Table 13–1 shows the definitions that are dumped in each case.

Table 13–1: Definitions Dumped To the Definition File

Definitions	All Tables	Selected Tables
Tables, fields, and indexes	Yes	Yes
Sequence definitions	Yes	No
Auto-connect records	Yes	No
Collation	No	No
_User	No	No

If you dump individual tables, you must also dump the sequence definitions and auto-connect records separately. For instructions, see the "Dumping Sequence Definitions" and "Dumping Auto-connect Records" sections later in this chapter.

13.2.1 Definition File Format

When you dump definitions, the Data Dictionary or the Data Administration tool creates a data definitions file. The Data tool then uses the data definitions file to load table definitions into the database. Figure 13–1 shows a sample data definitions file for a single table.

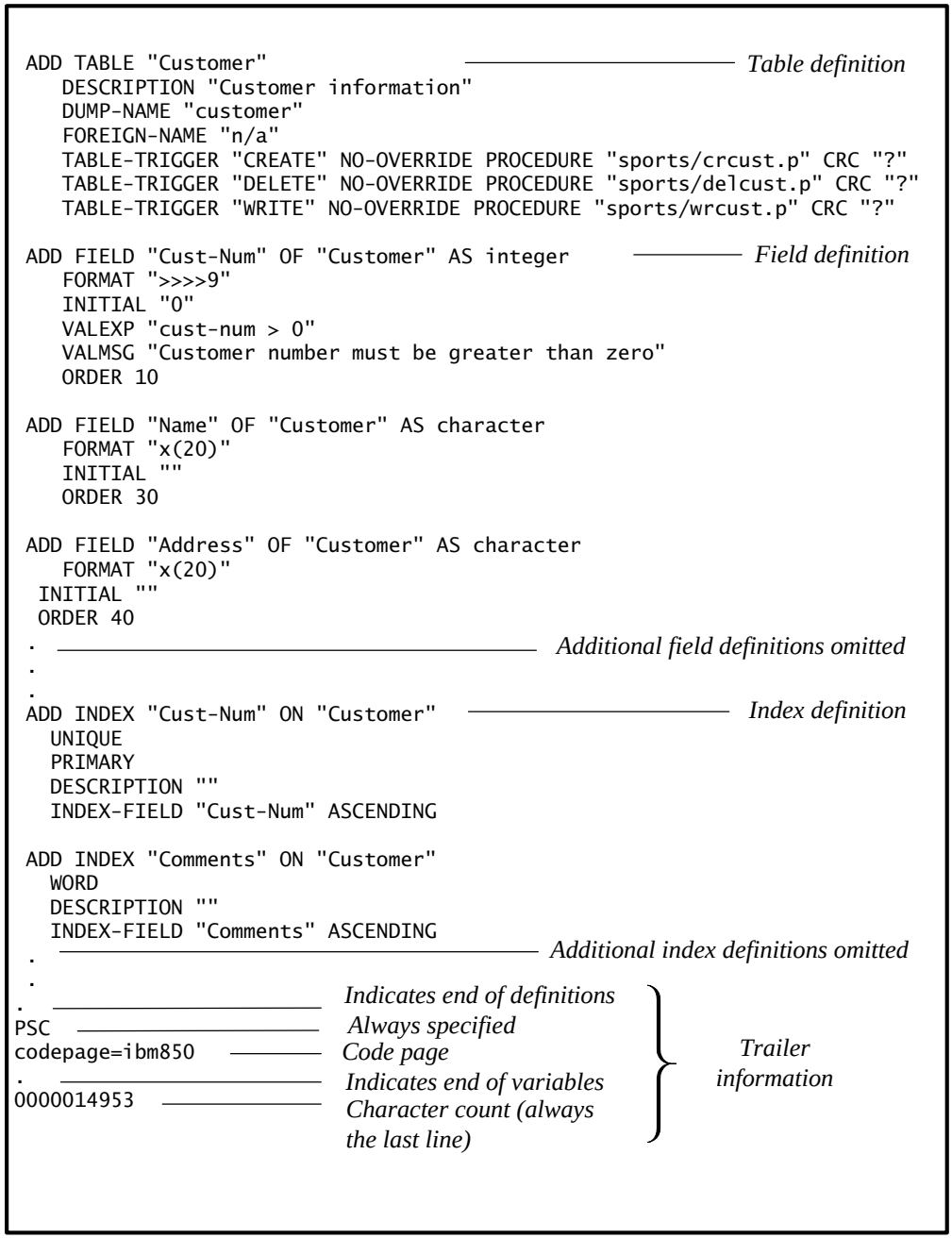


Figure 13–1: Sample Data Definitions File

If you modify a data definitions file, tables, fields, and indexes can be intermixed within one CREATE DATABASE section. However, be sure of the following:

- Define a file before any fields contained in that file.
- Define a field before any index using that field.
- Specify information using the correct syntax.
- All definitions must belong with the most recent ADD DATABASE or CREATE DATABASE entry in the DF file.

The trailer information contains the values used by the database definitions. If you edit the definitions file or create one manually, be sure to specify these values correctly. Table 13–2 explains these values.

Table 13–2: Data Definitions File Trailer Values

Field	Description
codepage=*codepage*	The code page used by the database. For more information, see the *Progress Internationalization Guide*.
Character count	The number of characters contained in the contents file, up to and including the period that indicates the end of the contents. This number is always a 10-digit number with leading zeros, if necessary.

13.2.2 Dumping 4GL Definitions

Follow these steps to dump database or table definitions:

1 ♦ Access the Data Administration tool. The Data Administration main window appears.

2 ♦ Make sure you are using the database (the source) from which you want to dump the table definitions.

3 ♦ Choose Admin→ Dump Data and Definitions→ Data Definitions (.df file). The Data Administration tool lists all the tables defined for the database alphabetically.

4 ♦ Choose the table where you want to dump the definitions, or choose ALL.

If you choose ALL, the DF file will also contain the sequences and auto-connect record definitions, but not the collation/conversion table. The Data Administration tool displays a default name for the file that you can dump data definitions into (hidden tables are not dumped). This default file is always the name of the table or database with a .df extension. The Data Administration tool truncates table names to eight characters. When you dump only one table, the table dump name becomes the default for its corresponding contents dump file. For example, if you specify `customer.d` as the filename, when you dump the file contents, the default filename is `customer.d`. If you specify to dump all the tables, the default name is *`db-name.d`*.

5 ♦ Accept this default or enter a different name and choose OK. The Data Administration tool displays each object name as it writes its definition to the data definitions file.

You see each table name on the screen as the Data Administration tool writes its definition to the DF file. After all of the table definitions are dumped, the Data Administration tool displays a status message and prompts you to continue.

6 ♦ Choose OK to return to the Data Administration main window.

13.2.3 Creating an Incremental 4GL Data Definitions File

An incremental data definitions file is a data definitions file that contains the schema differences between two databases. When you load the incremental data definitions file into a target database, the target database will then have the same schema as the source database. For example, you might use this to upgrade a production database to the same schema as a development database.

NOTE: If the data type of the extent of a field has changed, the Data tool writes a DROP statement to the .df file to delete the existing field, and then an ADD statement to re-create the field with its new definition. When the field is deleted from the database, the contents of that field are also deleted.

To save the contents of the field, save the data before running this procedure, or rename the old field, create a new field separately, and write a Progress 4GL procedure that transfers the data from the old field to the new field. Then use the incremental dump procedure to update the rest of the schema, including deleting the obsolete old field.

Follow these steps to create an incremental data definitions file:

1 ♦ Access the appropriate Data tool (the Data Administration tool if you are using a graphical interface or the Data Dictionary if you are using a character interface).

2 ♦ Select the source database as the current working database. The source database is the database that has the schema that you want the definitions file to have.

3 ♦ Choose Admin→ Dump Data and Definitions→ Create Incremental .df File. The Data tool lists all the connected databases.

4 ♦ If you have more than two databases connected, choose the target database. The target database is the database to update.

The Data tool prompts you for the file where you want to write the differences.

5 ♦ Specify the file where you want to write the differences. The default filename is `delta.df`. The Data tool displays the file, field, sequence, and index names as it compares the databases.

6 ♦ After comparing the database, the Data Administration tool or the Data Dictionary returns you to the main window.

NOTE: If you use this option to create a DF file in conjunction with r-code files to update schema changes, you must load the DF file and recompile before you can run the new r-code. You must recompile because the Data tool reorders the indexes during the dump and load procedure.

13.2.4 Dumping Sequence Definitions

You can dump sequence definitions and values using either Data tool. The Data tool stores the sequence definitions in the full database schema definition file that you name, and stores the sequence values in a separate `_seqvals.d` file (or the file you name). You can dump the sequence definitions and values separately.

Follow these steps to dump sequence definitions:

1 ♦ Access the appropriate Data tool (the Data Administration tool if you are using a graphical interface or the Data Dictionary if you are using a character interface).

2 ♦ Make sure that the database containing the sequences you want to dump or load is the current working database.

3 ♦ Choose Admin→ Dump Data and Definitions→ Sequence Definitions. The Data tool prompts you for the file where you want to write the sequence definitions. The default filename is `_seqdefs.df`.

4 ♦ Specify the filename or use the default value. After dumping the sequence definitions, the Data tool displays a status message and prompts you to continue.

5 ♦ Choose OK to return to the Data Administration or Data Dictionary main window.

13.2.5 Dumping Auto-connect Records

Follow these steps to dump auto-connect records:

1 ♦ Access the appropriate Data tool (the Data Administration tool if you are using a graphical interface or the Data Dictionary if you are using a character interface).

2 ♦ Choose Admin→ Dump Data and Definitions→ Auto-connect Records Only. The data tool prompts you for the output file where you want to write the auto-connect records. The default filename is `_auto.df`.

3 ♦ Specify a new filename or accept the default. After dumping the auto-connect records, the Data tool displays a status message and prompts you to continue.

4 ♦ Choose OK to return to the Data Administration or Data Dictionary main window.

13.3 Dumping Database Contents

When you dump database contents, you can dump the following:

- Table contents
- Sequence values
- User table contents
- SQL View File contents

NOTE: For a complete description of dumping and loading SQL-92 content, see Chapter 19, "Database Administration Utilities."

Progress provides two methods of dumping database contents: you can use the PROUTIL command to dump the data in binary format, or you can use the Data Administration tool user interface to dump the data in text format. Dumping the data in binary format with the PROUTIL command is faster.

13.3.1 Dumping Table Contents With PROUTIL

Use the following PROUTIL command to perform a binary dump:

```
proutil db-name -C dump [owner-name.]table-name directory [-index num]
```

db-name

Specifies the database from which you want to dump. You must completely define the path.

owner-name

Specifies the owner of the table containing the data you want to dump. You must specify an owner name unless the table's name is unique within the database, or the table is owned by PUB. By default, Progress 4GL tables are owned by PUB.

table-name

Specifies the name of the table containing the data you want to dump.

directory

Specifies the name of the target directory where the data will be dumped.

[-index num]

Specifies an index to use to dump the table's contents. If you choose not to use this option, the command uses the primary index to dump the table.

You can use the PROUTIL IDXANALYS utility to help determine what the index number is. For the complete syntax of the PROUTIL IDXANALYS utility, see Chapter 19, "Database Administration Utilities."

Results Of a Binary Dump With PROUTIL

PROUTIL DUMP writes data from a table to a dump file. The name of the resulting dump file depends on the owner of the table. By default, Progress 4GL tables are owned by PUB. When tables owned by PUB are dumped to a file, the filename is the table name with `.bd` appended. For example, *tablename.bd*.

However, when tables owned by anyone other than PUB are dumped to a file, the resulting filename contains the owner name and table name. For example, *ownername_tablename.bd*.

On UNIX systems that have a 2GB file size limitation (Alpha OSF does not), PROUTIL DUMP creates multiple files when you dump a table larger than 2GB. For example, when you dump data from a table with the name "customer" that is 6.4GB, PROUTIL DUMP creates four binary dump files: `customer.bd`, `customer.bd2`, and `customer.bd3`, each of which is approximately 2GB, and `customer.bd4`, which is approximately 0.4GB. The PROUTIL DUMP procedure adds header blocks to the binary dump files. As a result, the total size of the binary dump files is slightly larger than the table itself.

On Windows NT and Alpha OSF, however, there is no 2GB file size limitation. On Windows NT and Alpha OSF, PROUTIL DUMP creates only one binary dump file regardless of the size of the table.

Format Of a Binary Dump File

Each binary dump file contains a header and a description for each record in the table. The dump file appears in the following format:

Header	Record length	Table number	Binary record	Record CRC

The file header contains information that appears in this order:

1. Version number
2. Date and time the file was created
3. Name of the table being dumped
4. Number of the table being dumped
5. CRC of the table being dumped
6. Number of fields in the table
7. Name of the database where the table resides
8. Section number
9. Number of the first record
10. Table owner

Section numbers that appear in the file header correspond to a binary dump file created by PROUTIL DUMP when the table was larger than the UNIX 2GB file size limitation. (See the "Results Of a Binary Dump With PROUTIL" section.) For example, Section 1 corresponds to the binary dump file named `customer.bd`, Section 2 corresponds to `customer.bd2`, Section 3 corresponds to `customer.bd3`, and Section 4 corresponds to the `customer.bd4` binary dump file.

NOTE: To support the dump and load of binary large objects (BLOBS), PROUTIL DUMP adds more items to the header of the binary dump file.

13.3.2 Dumping Field Contents With PROUTIL

Use the following syntax to perform a selective binary dump:

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C dumpspecified` [*owner-name.*] *table-name.field-name operator field-value directory*

db-name

Specifies the database from which the dump will occur. You must completely define the path.

owner-name

Specifies the owner of the table containing the data you want to dump. You must specify an owner name unless the table's name is unique within the database, or the table is owned by "PUB." By default, Progress 4GL tables are owned by PUB.

table-name

Specifies the name of the table containing the data you want to dump.

field-name

Specifies the name of the field containing the data you want to dump.

operator

Specifies the operator to be used: EQ (equals to), GT (greater than), GE (greater than or equal), LT (less than), or LE (less than or equal).

field-value

Specifies the value against which the filed contents will be compared.

directory

Specifies the name of the target directory where the data will be dumped. A directory must be specified.

EXAMPLES

The following syntax dumps all order date values greater than 2/3/02 from the Sports2000 database:

```
proutil sports2000 -C dumpspecified order.order_date GT '2/3/02'
```

The following syntax dumps all item prices less than $25.90 from the Sports2000 database:

```
proutil sports2000 -C dumpspecified item.price LT 25.9
```

The following syntax dumps all order ship dates of 12/24/01 from the Sports2000 database into the /dev/results directory:

```
proutil sports2000 -C dumpspecified order.ship_date EQ '12/24/01'
/dev/results
```

The following syntax dumps all back ordered items from the Sports2000 database into the /inventory/warehouse1 directory:

```
proutil sports2000 -C dumpspecified order_line.backorder EQ yes
/inventory/warehouse1
```

For more information on selective binary dumps, see the "PROUTIL DUMPSPECIFIED" section of Chapter 19, "Database Administration Utilities."

13.3.3 Dumping Table Contents With a Data Tool

When you dump table data, the Data tool creates a contents file that contains data from each table you specify. This file has a .d extension. The Data tool uses the contents file to load data into the database. The Data tool creates a separate contents file for every table you dump.

Follow these steps to dump table contents:

1 ♦ Access the appropriate Data tool (Data Administration tool if you are using a graphical interface or the Data Dictionary if you are using a character interface).

2 ♦ Make sure you are using the database (the source) where you want to dump the table data.

3 ♦ Choose Admin→ Dump Data and Definitions→ Table Contents (.d files). The Data tool lists all the tables defined for the database alphabetically.

4 ♦ Mark the table contents you want to copy. You can mark tables with asterisks (*), then use the Select Some and Deselect Some buttons to select or deselect groups of tables. If you are using a character interface, use **F5** and **F6**.

When you dump a single table, the Data tool displays a default name for the file that you can dump the table contents into (hidden tables are not dumped). This default file is always the dump name of the table definition file, with a .d extension.

If you want to specify a file other than the default file, type the name in the Write output to file field. You can use filenames longer than eight characters for the data tables if your operating system allows it, but the Data tool requires a unique eight-character dump name (which you supply when you create the table).

When you dump table contents for an entire database, the Data tool prompts you for a directory name to write the contents files to. If you do not specify a directory, the Data tool dumps the files into the current directory. The Data tool names each contents file with its corresponding table name.

5 ♦ Accept the default or enter a different name.

6 ♦ If you want to use character mapping, enter the character mapping, then choose OK. See the *Progress Internationalization Guide* for information about character mapping, PROTERMCAP, and national language support.

The Data tool displays each table name as it writes the table contents to the table contents file. After dumping the table contents, the Data tool displays a status message and prompts you to continue.

7 ♦ Choose OK to return to the Data Administration or Data Dictionary main window.

NOTE: You must dump the sequence values separately. See the "Dumping Sequence Values" section for information about dumping sequence values.

13.3.4 Contents File Format

Figure 13–2 shows a sample contents file.

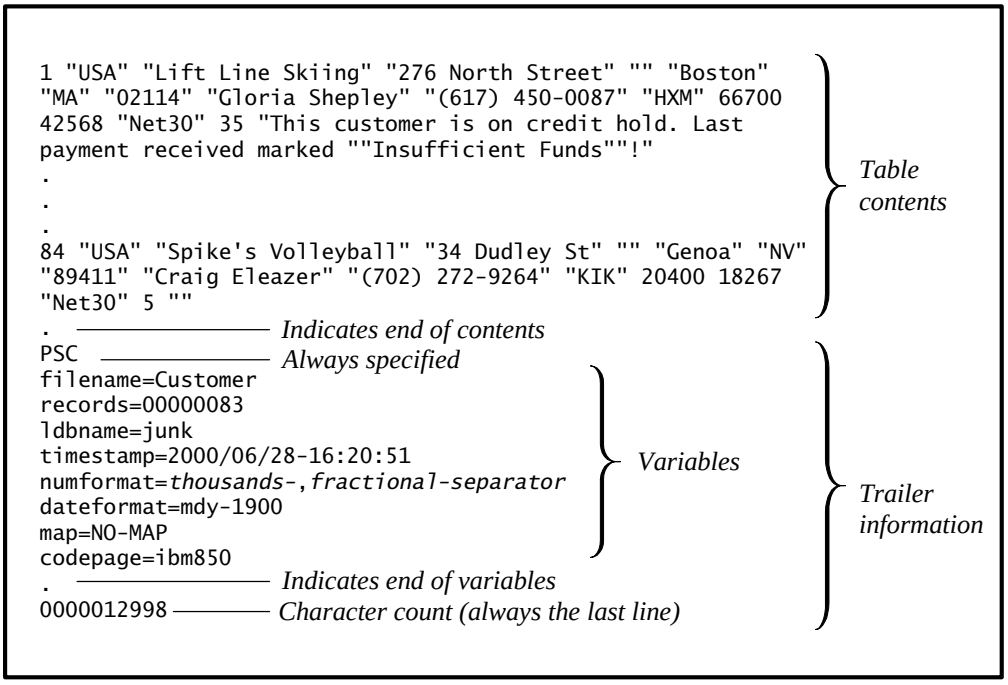


Figure 13–2: Sample Contents File

The trailer information contains information about the source database and certain startup parameters specified for the session in which the contents file was created. Certain variables are included for informational purposes only; other variables are used to load the data correctly. If you edit the contents (.d) file or create one manually, be sure to specify these variables correctly.

Table 13–3 explains these variables. If the database uses the default values for these variables, these variables do not appear in the contents file.

Table 13–3: Contents File Trailer Variables

Field	Description
filename=*table-name*	Table name.
records=*num-records*[1]	Number of records contained in the file.
ldbname=*logical-db-name*	Logical database name of the source database.
time stamp=*time stamp*	Time stamp of the database.
num format=[1] *thousands-separator*, *fractional-separator*	The numeric value of the character that represents the thousands-separator and fractional-separator identified in the code page. When you attempt to load the table contents file, the session and file must use the same numeric format. If not, the Progress 4GL reports a run-time error.
dateformat=*date-format*[1]	The date format used by the database contents.
map={MAP *protermcap-entry* | NO-MAP}[1]	The character mapping to support the MAP option on the INPUT FROM and OUTPUT TO statements. You can specify a mapping between a standard set of ASCII (7-bit) characters and extended (8-bit) characters in the PROTERMCAP file. Progress uses the PROTERMCAP entries to build a translation table for the stream. MAP specifies an entry that has been defined in the PROTERMCAP file. NO-MAP directs Progress to bypass character translation altogether.
codepage=*codepage*	The code page used by the database. For more information, see the *Progress Internationalization Guide*.
Character count	The number of characters contained in the contents file, up to and including the period (.) that indicates the end of the contents. This number is always a ten-digit number with leading zeros, if necessary.

[1] Information used when loading the file. If the value is specified in the trailer and is specified incorrectly, the data will not load.

13.3.5 Dumping Sequence Values

Follow these steps to dump sequence values:

1 ♦ Access the appropriate Data tool (the Data Administration tool if you are using a graphical interface or the Data Dictionary if you are using a character interface).

2 ♦ Make sure that the database containing the sequences you want to dump or load is the current working database.

3 ♦ Choose Admin→ Dump Data and Definitions→ Sequence Values. The Data tool prompts you for the file you want to write the sequence values to. The default filename is `_seqvals.d`.

4 ♦ Specify the filename or use the default value. After dumping the sequence values, the Data tool displays a status message and prompts you to continue.

5 ♦ Choose OK to return to the Data Administration or Data Dictionary main window.

13.3.6 Dumping User Table Contents

Follow these steps to dump the user table contents:

1 ♦ Access the appropriate Data tool (the Data Administration tool if you are using a graphical interface or the Data Dictionary if you are using a character interface).

2 ♦ Make sure you are using the database from which you want to dump the table data.

3 ♦ Choose Admin→ Dump Data and Definitions→ User Table Contents. The Data tool prompts you for the file to write the user file contents to. The default filename is `_user.d`.

4 ♦ Specify the filename or accept the default. After dumping the user file contents to the specified file, the Data tool displays a status message and prompts you to continue.

5 ♦ Choose OK to return to the Data Administration or Data Dictionary main window.

13.3.7 Dumping an SQL View File's Contents

Follow these steps to dump an SQL view file's contents:

1 ♦ Access the appropriate Data tool (the Data Administration tool if you are using a graphical interface or the Data Dictionary if you are using a character interface).

2 ♦ Make sure you are using the database (the source) where you want to dump the table data.

3 ♦ Choose Admin→ Dump Data and Definitions→ Views. The Data tool prompts you for the file, then for the file that you want to write the user file contents to. The default filename is `_view.d`.

4 ♦ Specify the filename or accept the default. After dumping the view file contents to the specified file, the Data tool displays a status message and prompts you to continue.

5 ♦ Choose OK to return to the Data Administration or Data Dictionary main window.

NOTE: For a complete description of dumping and loading SQL-92 content, see Chapter 19, "Database Administration Utilities."

13.4 Loading Database Definitions

You can load either all the data definitions for a table or only those definitions that have changed.

13.4.1 Loading Table Definitions

The Data tools use a data definitions file that you specify to load table definitions into the database. A data definitions file contains definitions for tables, fields, and indexes.

Follow these steps to load table definitions into a database:

1 ♦ Access the appropriate Data tool (the Data Administration tool if you are using a graphical interface or the Data Dictionary if you are using a character interface).

2 ♦ Make sure you are using the database (the target) that you want to load the table definitions into.

3 ♦ Choose Admin→ Load Data and Definitions→ Data Definitions (.df files). The Data tool prompts you for the name of the file that contains the data definitions you want to load into the current database. The default filename is the logical name of the current working database, with a .df extension.

4 ♦ Specify a file or accept the default.

5 ♦ Specify whether you want to stop the load as soon as the Data tool encounters a bad definition statement. The Data tool displays each item as it loads the definitions for that object. The Data tool displays a status message and prompts you to continue. If you choose not to stop on errors, the load will continue with the next entry.

NOTE: Whatever you choose, if the Data tool encounters any error, it backs out any loaded definitions.

6 ♦ Choose OK to return to the Data Administration or Data Dictionary main window.

The database now contains the table, field, index, or sequence, but none of the data.

13.4.2 Loading Updated 4GL Data Definitions

Follow these steps to update an existing (production) database schema to include schema changes made in a new (development) database. Note that this procedure can also be used to merge two databases:

1 ♦ Make a copy of the database you want to update and save the original. The database should not be empty.

2 ♦ Enter the database that includes the new, modified data definitions.

3 ♦ Access the appropriate Data tool (the Data Administration tool if you are using a graphical interface or the Data Dictionary if you are using a character interface).

4 ♦ Choose Database→ Connect Database. The Database Connect dialog box appears.

5 ♦ Enter the name of the database you want to connect to and choose OK. The Data tool connects the database and returns you to the Data tool's main window.

6 ♦ Choose Admin→ Dump Data and Definitions→ Create Incremental .df File. The Create Incremental .df File dialog box appears.

The Create Incremental .df File option compares the data definitions in the nonempty copy to the current database schema and creates a new data definitions file. The new DF file contains a record for each difference between the two schemas. The differences include any added, renamed, changed, or deleted table, field, or index.

If a table, field, or index exists in the old database but not in the new schema, Progress asks if you renamed the object. If you answer no, a record appears in the new DF file marking the object as deleted.

If the new schema includes a new, unique, active index, the Data tool prompts you to deactivate it. If you do not deactivate the index and there are duplicate keys in the old database, the system aborts your attempt to load new definitions into the old database. If you deactivate the index, the load procedure defines the new index but does not create the index file. You must complete Step 8 to build and activate the index after loading the new data definitions.

7 ♦ Enter the database name or accept the default databases, then choose OK.

8 ♦ Connect to the copy of the old database.

9 ♦ Load the updated data definitions by choosing Admin→ Load Data and Definitions→ Data Definitions (.df files).

10 ♦ If you deactivated any indexes in Step 4, re-create data in the indexed fields as required to avoid duplicate keys, then reactivate the indexes with PROUTIL IDXBUILD.

11 ♦ The Data tool updates the old database schema to match the modified schema. Compile and test all your procedures against the updated database.

13.5 Loading Database Contents

These sections describe how to load the following types of contents:

- Table contents
- User table contents
- SQL file view contents
- Sequence values

NOTE: For a complete description of dumping and loading SQL-92 contents, see Chapter 19, “Database Administration Utilities.”

Progress provides three methods of loading table contents. You can use the PROUTIL command to load the data in binary format, the Data Administration tool’s user interface to load the data in text format, or the PROUTIL command with the Bulk Loader (BULKLOAD) qualifier. You can perform a binary load only on database contents that were created with a binary dump.

13.5.1 Loading Table Contents With the PROUTIL Command

Use the following PROUTIL command to perform a binary load:

```
proutil db-name -C load filename
```

db-name

Specifies the database where you want to load the data. You must completely define the path.

filename

Specifies the binary dump file that you want to load. You must completely define the path.

To load multiple binary dump files into the target database, specify each file individually. For example:

```
proutil db-name -C load customer.bd
proutil db-name -C load customer.bd2
proutil db-name -C load customer.bd3
proutil db-name -C load customer.bd4
```

When the load procedure finishes, it reports the number of records that were loaded.

NOTE: To update the index entries in the database when performing a binary load, use the BUILD INDEXES qualifier. For more information on the BUILD INDEXES qualifier, see the "PROUTIL LOAD Qualifier" section in Chapter 19, "Database Administration Utilities."

Recovering From an Aborted Binary Load

The procedure to recover from an aborted binary load depends on how much the binary load completed before the abort. If the binary load got all the rows loaded and aborted during the sort phase, then recovery entails running the index rebuild utility on the effected table. If the binary load aborted during the load of the rows into the database follow these steps:

1 ♦ Truncate the storage area.

2 ♦ Restart the load operation.

You can find out what phase the binary load completed by examining the database log (.lg) file. In this file, you will see messages identifying the major phases of the load operation, such as:

- Binary load session started
- Record load phase started
- Record load phase completed
- Starting index build phase
- Index build phase completed
- Binary load session ended

13.5.2 Loading Table Contents With a Data Tool

Progress uses a contents file that you specify to load table data into the database. A contents file contains all the data for a table.

NOTE: Table definitions must be in the database before you can load table contents.

There are two ways to load the table contents:

- Use the appropriate Data tool (the Data Administration tool if you are using a graphical interface or the Data Dictionary if you are using a character interface).
- Use the Bulk Loader (BULKLOAD) qualifier of the PROUTIL utility. For information on using the Bulk Loader, see the "Bulk Loading" section later in this chapter.

Follow these steps to load table contents into a database:

1 ♦ Access the appropriate Data tool (the Data Administration tool if you are using a graphical interface or the Data Dictionary if you are using a character interface).

2 ♦ Make sure that the working database is the target database where you want to load the table contents.

3 ♦ Choose Admin→ Load Data and Definitions→ Table Contents (.d files). The Data tool alphabetically lists all the tables defined for the database.

4 ♦ Mark the tables whose contents you want to copy. Use the Select Some and Deselect Some buttons to select or deselect groups of tables. For a character interface, use **F5** and **F6** to select table names.

The Data tool prompts you for the name of the contents file — or the directory that contains the contents files — you want to load into the current database.

5 ♦ Specify the directory name, filename, or use the default value.

When you load the contents of an entire database, enter a directory where you want to load the contents files. If you do not specify a directory, the Data tool loads the files from the current directory. The Data tool loads each table from the corresponding *table-dumpname.d* filename. If you choose a single table to load, the Data tool displays a default name for the file where you can load the table contents. This default file is always the name of the table definition file, with a .df extension.

The Data tool also prompts you for an acceptable error rate.

6 ♦ Specify the error rate. As the Data tool loads records from any files you designate, it might encounter data that cannot be loaded. For example, a record might have data in one of its fields that is too long for that field's defined format. The Data tool does not load the record. If you specify an error rate of 10 percent, the Data tool must successfully load 90 records from every set of 100 records loaded. If you use the graphical interface to load table contents, you can choose to output the errors to a file or output the errors to the screen. After loading the table contents, the Data tool prompts you to continue.

7 ♦ Choose OK to return to the Data Administration or Data Dictionary main window.

NOTE: You must load the sequence values separately. See the "Loading Sequence Values" section for information about loading sequence values.

13.5.3 Loading User Table Contents

Follow these steps to load the user table contents:

1 ♦ Access the appropriate Data tool (the Data Administration tool if you are using a graphical interface, or the Data Dictionary if you are using a character interface).

2 ♦ Choose Admin→ Load Data and Definitions→ User Table Contents. The Data tool prompts you for the file from where you want to read the user file contents. The default filename is `_user.d`.

3 ♦ Specify the filename or accept the default. After loading the user file contents to the specified file, the Data tool displays a status message and prompts you to continue.

4 ♦ Choose OK to return to the Data Administration or Data Dictionary main window.

13.5.4 Loading an SQL View File Contents

Follow these steps to load an SQL view file's contents:

1 ♦ Access the appropriate Data tool (the Data Administration tool if you are using a graphical interface or the Data Dictionary if you are using a character interface.

2 ♦ Choose Admin→ Load Data and Definitions→ SQL Views. The Data tool prompts you for an input file from which to load the SQL views. The default filename is `_view.d`.

3 ♦ Specify the filename or accept the default. After loading the view file contents from the specified file, the Data tool displays a status message and prompts you to continue.

4 ♦ Choose OK to return to the Data Administration or Data Dictionary main window.

NOTE: For a complete description of dumping and loading SQL-92 content, see Chapter 19, "Database Administration Utilities."

13.5.5 Loading Sequence Values

The Data Administration tool and the Data Dictionary use a contents file that you specify to load sequence values into the database. Usually, this file is called `_seqvals.d` file.

Follow these steps to load sequence values into a database:

1 ♦ Access the appropriate Data tool (the Data Administration tool if you are using a graphical interface or the Data Dictionary if you are using a character interface).

2 ♦ Make sure that the working database is the target database where you want to load the table contents.

3 ♦ Choose Admin→ Load Data and Definitions→ Sequence Current Values. The Data tool prompts you for the filename where you want to write the sequence values. The default filename is `_seqvals.d`.

4 ♦ Specify the filename or use the default value. After the sequence values are loaded, the Data tool displays a status message and prompts you to continue.

5 ♦ Choose OK to return to the Data Administration or Data Dictionary main window.

13.6 Bulk Loading

The Bulk Loader utility loads data at a higher speed than the Load utility provided with the Data Dictionary or Data Administration tool.

NOTE: The bulk loader works only with Progress databases. Some non-Progress databases offer a similar bulk loading tool. If you are using a non-Progress database and a bulk loading tool is not available, you must use the standard load utility in the Data Administration tool or Data Dictionary.

13.6.1 Creating a Bulk Loader Description File

You must create a bulk loader description file before you can run the Bulk Loader utility. You use the Data Administration tool or the Data Dictionary to create the description file. The default filename is *table-dumpname*.fd. If you choose more than one table, the Data tool adds the extension .fd to the name of the database. It then places all of the bulk loader information in that one file.

NOTE: For information on how to dump table data into contents files, see the "Dumping User Table Contents" section.

Follow these steps to create a bulk loader description file:

1 ♦ Access the appropriate Data tool (the Data Administration tool if you are using a graphical interface or the Data Dictionary if you are using a character interface).

2 ♦ Choose Admin→ Create Bulk Loader Description File. The Data tool alphabetically lists all the tables defined for the database.

3 ♦ Select the tables for which you want to create bulk loader description files. Use the Select Some and Deselect Some buttons to select or deselect groups of tables. For a character interface, use **F5** and **F6**. The Data tool prompts you for the bulk load filename. The default filename is *table-dumpname*.fd.

4 ♦ Specify the filename or accept the default. After creating the file, the Data tool displays a status message and prompts you to continue.

5 ♦ Choose OK to return to the Data Administration or Data Dictionary main window.

13.6.2 Modifying a Bulk Loader Description File

You can use a text editor to change a Bulk Loader description (FD) file. Table 13–4 lists the tasks and actions.

Table 13–4: Modifying the Description File

Task	Action
Include comments in the FD file.	Place a pound sign (#) at the beginning of each line.
Include the description of multiple tables in one description file.	Separate tables with a period on a line by itself.
Skip a field.	Replace the field name with a caret (^).

Using a text editor, change the description file to appear as follows:

```
customer
  Cust-num
  Name
  ^
  ^
  City
  St
  ^
  Phone
```

In this example, the Bulk Loader utility loads the Cust-num, Name, City, St, and Phone fields, since they are the only fields specified in the description file. The carat (^) directs the Bulk Loader to skip a field. The Bulk Loader does not load any of the fields after Phone because they do not appear in the description file. The customer table name appears by itself on the top line.

By default, the Bulk Loader adds .d and .e extensions to the table name to produce the two file names it requires to operate. It assumes that both of the files (the data file and the error file) begin with the table name. If this assumption is false, you must specify the different filenames. For example, if you dump the customer table into `cust.d` instead of `customer.d`, you must specify the name `cust.d` in the bulk loader description file. Otherwise, the Bulk Loader searches the current directory for `customer.d`:

```
#This is an example
customer   cust.d cust.e
  Cust-num
  Name
  ^
  ^
   City
  St
  ^
  Phone
.
item
  Item-num
  Idesc
.
order
  Order-num
  ^
  Name
```

13.6.3 Loading Table Contents With the Bulk Loader Qualifier

PROUTIL with the BULKLOAD qualifier allows you to load data tables into a database. Using contents files and bulk loader description (.fd) files, the Bulk Loader loads large amounts of data faster than the regular Data Administration or Data Dictionary Load utility.

The Bulk Loader can also load table data in formats other than contents files. For information about other file formats PROUTIL can load, see the IMPORT Statement in the *Progress Programming Handbook*.

The Bulk Loader bypasses any CREATE TRIGGER statements and deactivates all indexes it encounters.

Follow these steps to load table contents with the Bulk Loader utility:

1 ♦ Create a Bulk Loader description file using the Data Administration tool, Data Dictionary, or a text editor. If you use the Data Dictionary or Data Administration tool, it automatically writes the description file. If you use a text editor, you must create the file.

Using a text editor to modify a description file that the Bulk Loader has created allows you to customize the file to suit your needs. For more information see the "Creating a Bulk Loader Description File" section.

2 ♦ Verify that the table definitions are in the database.

3 ♦ Run the Bulk Loader utility using the PROUTIL BULKLOAD command:

```
proutil db-name [ -yy n ] -C bulkload fd-file [ -Bn ]
```

db-name

Specifies the database you are using.

-yy n

Indicates the start of a 100-year period iin which any two-digit DATE value is defined; *n* specifies a four-digit year (1900, for example) that determines the start of the 100-year period. The default is 1950, but the -yy n value must match the -yy n value used when the data was dumped.

fd-file

Identifies the bulk loader description file you are using.

-Bn

Blocks in DataBase Buffers startup parameter; n specifies the number of blocks.

See Chapter 19, "Database Administration Utilities," for information about the PROUTIL utility and the BULKLOAD qualifier.

NOTE: On a minimally configured PC, you might encounter memory problems when using the Bulk Loader utility. To work around these problems, split the description file into several smaller files.

Figure 13–3 shows a description file created with the Data Administration tool.

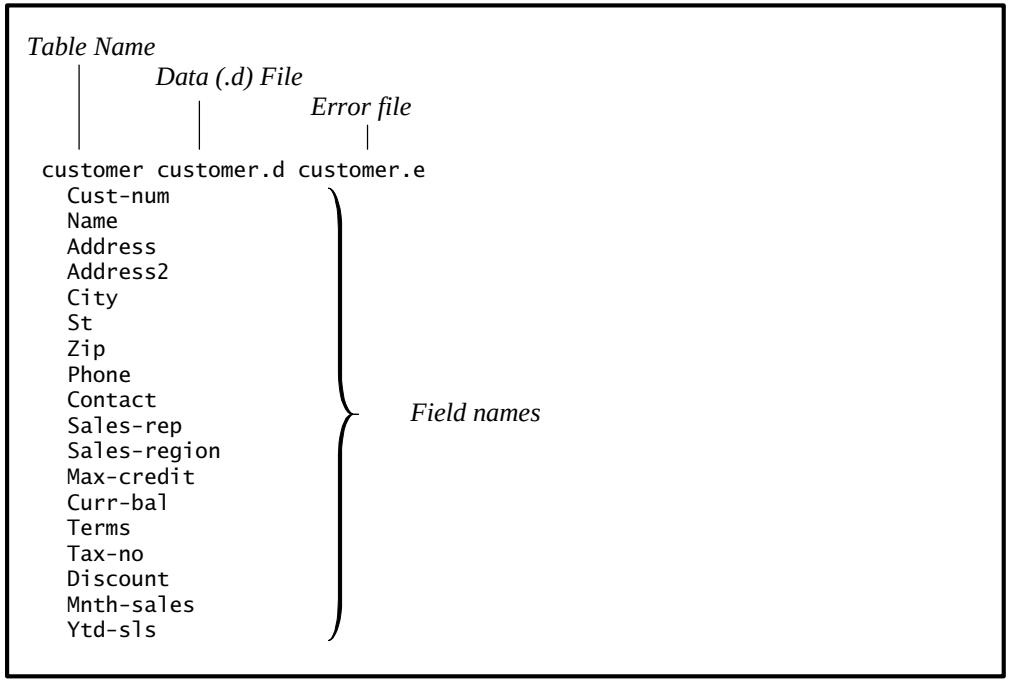


Figure 13–3: Dictionary-created Description File Example

In this example, customer is the name of the database table, `customer.d` is the name of the data file you want to load, and `customer.e` is the name of an error file that the Bulk Loader creates if errors occur during the bulk load. The field names in the customer table are Cust-num, Name, Address, etc.

The Bulk Loader utility checks the description file to determine which data file contains the customer table's dumped records. In this example, it is `customer.d`.

The order of the fields in the description file must match the order of the fields in the data (.d) file. If they do not match, the Bulk Loader attempts to load data into the wrong fields.

The Bulk Loader utility automatically deactivates a data table's indexes, so you must run the Index Rebuild utility after the Bulk Loader loads the data files. If the Bulk Loader encounters duplicate key values during a load, it continues to load the data file's records. You must manually correct the data before you run the Index Rebuild utility.

13.7 Reconstructing Bad Load Records

If the Database Administrator tool or the Data Dictionary encounters errors while loading a data file, it creates an error file. You can use the error file and the original data file to build a new data file for all the bad records.

Follow these steps to build this new data file:

1 ♦ Access the appropriate Data tool (the Data Administration tool if you are using a graphical interface or the Data Dictionary if you are using a character interface).

2 ♦ Choose Admin→ Load Data and Definitions→ Reconstruct Bad Load Records. The Reconstruct Bad Load Records dialog box appears.

3 ♦ Specify the original data file, the error file, and the new output file, then choose OK. The default filename for the new data file is `error.d`. After writing the new output file, the Data tool displays a status message and prompts you to continue.

4 ♦ Choose OK to return to the Data Administration or Data Dictionary main window.

5 ♦ Use a text editor to edit the new output file and fix the bad records after the Data tool builds the file. Once you have fixed the records, you can reload the file.

13.8 Specialized Dump and Load Techniques

The following sections provide step-by-step instructions for dumping and loading data for specialized purposes.

13.8.1 Creating a Starting Version Of a Database

Suppose you finish writing an application, and your database contains not only the definitions users require to start running the application, but also the data you used to test the procedures. Before you put the application to use, you want to create a version of the database that contains only data definitions and no data. This database becomes a template you can copy for testing purposes, for your own use, and for distribution. Figure 13–4 illustrates how the Progress Database Utility (PRODB) copies the starting version of a database.

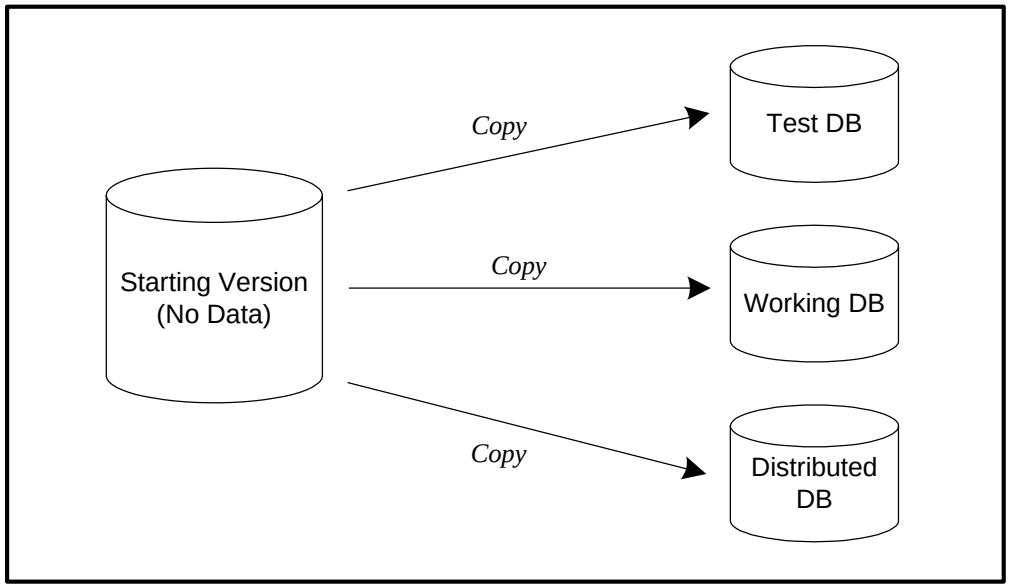


Figure 13–4: Copying the Starting Version Of a Database

To create a new database that contains table definitions from another database, use the Dump facility to dump the table definitions, the PRODB utility to create the new database, and the Load facility to load the table definitions into the new database.

Follow these steps to create a starting version of a database:

1 ♦ Start a Progress session with the database (the source) you want to copy.

2 ♦ Dump the database table definitions into a data definitions file.

3 ♦ Dump the database table data into a contents file.

4 ♦ Create a new copy of the empty database.

5 ♦ Start a Progress session with the new database (the target) you created in Step 4.

6 ♦ With the data definitions file you created in Step 2, load the database definitions into the new database.

13.8.2 Using a Constant Table In Multiple Databases

You can use the Progress Dump and Load utility to copy an existing table from one database into another. For example, instead of writing a procedure to input state data into the state table in a new database, you can dump the table contents from the state table into a copy of the sports database, and then load those records into the state table in another database.

Follow these steps:

1 ♦ Start a Progress session with the database (the source) that has the table you want to copy.

2 ♦ Dump the contents of the table you want to copy into a contents file.

3 ♦ Return to the Progress Procedure Editor.

4 ♦ Switch from the current working database to the target database.

5 ♦ With the target database as the working database, load the contents of the table.

13.8.3 Economizing Disk Space

A database can inefficiently occupy system disk space over time, especially during development phases when frequent deletions of database elements occur. One benefit of dumping and reloading a database is better allocation of disk space.

Dumping and reloading a database involves essentially creating a new starting version of the database and loading the table contents into this new database. During the loading stage Progress repartitions the disk, removing any gaps created when database elements were deleted, thereby more efficiently allocating disk space. Use the PROUTIL utility with the TABANALYS qualifier to see how fragmented a database is. For information, see the description of the PROUTIL TABANALYS utility in Chapter 19, “Database Administration Utilities.”

Follow these steps to dump and reload a database:

1 ♦ Start a Progress session with the database (the source) you want to copy.

2 ♦ Dump the database table definitions into a data definitions file.

3 ♦ Dump the database table data into a contents file.

4 ♦ Dump the sequence values.

5 ♦ Create a new copy of the database.

6 ♦ Designate the new (target) database you created in Step 5 as the working database.

7 ♦ With the data definitions file you created in Step 2, load the database definitions into the new database.

8 ♦ Load the table contents files you created in Step 3 into the new database, using either the Progress Load utility or the Bulk Loader utility.

9 ♦ Load the sequence values.

10 ♦ Delete the old database using the PRODEL utility.

13.8.4 Optimizing Data For Sequential Access

If users repeatedly request data from the same tables in a determined order (for example, to create a certain report), dump and reload the data to optimize it for sequential access. Loading tables in order of average record size results in the most efficient organization of records for sequential access.

Follow these steps:

1 ♦ Use PROUTIL with the TABANALYS qualifier to determine the mean record size of each table in the database. See Chapter 19, "Database Administration Utilities," for a deatiled description of the PROUTIL TABANALYS qualifier.

2 ♦ Dump the definitions and data.

The Data Administration and Data Dictionary tools dump in order of primary index. If you access the records by another index, dump the data by that index. Use a Progress 4GL procedure similar to the following to dump the data by the index of your choice:

```
OUTPUT TO table-name.D.
FOR EACH table-name BY index
EXPORT table-name.
END
```

This procedure creates a contents (.d) file organized by order of access.

3 ♦ Load the tables of less than 1,000 bytes first, in order of average record size. Use the Data Dictionary or the Data Administration tool to load data one table at a time, or use the Bulk Loader utility and a description file to control the order.

   If you use the Bulk Loader, the order of the fields in the description file must match the order of the fields in the data file. If they do not match, the Bulk Loader attempts to load data into the wrong fields.

4 ♦ Load the remaining, larger tables, in order of average record size.

13.8.5 Optimizing Data For Random Access

If your application accesses data in a random order, for example an online transaction-processing application, you can dump and reload the data to optimize it for random access. Use a database with small, fixed-length extents spread across the volumes to balance the I/O across the disks. Use several clients to load data simultaneously, thereby spreading the data across all disks.

This technique also improves load performance on SMP platforms.

Follow these steps:

1 ♦ Dump the data and definitions.

2 ♦ Load the definitions using the Data Dictionary or the Data Administration tool.

3 ♦ In multi-user mode, start a server with a before-image writer (BIW) and asynchronous page writer (APW).

4 ♦ Start a client for each processor on your system. Have each client load certain tables. For each client, write a Progress 4GL procedure similar to the following:

```
/*Run this procedure after connecting to a database*/

DEFINE VAR RECCOUNT AS INT NO-UNDO.
DEFINE VAR NXT-STOP AS INT NO-UNDO.
DEFINE VAR i as INT NO-UNDO.

INPUT FROM tablename.d

TOP:
REPEAT TRANSACTION:

  NXT-STOP=RECCOUNT + 100
  REPEAT FOR table-name WHILE RECCOUNT < NXT-STOP
  ON ERROR UNDO, NEXT ON ENDKEY UNDO, LEAVE TOP:

    CREATE table-name.
    IMPORT table-name.
    RECCOUNT = RECCOUNT + 1.
  END.
END.
```

The clients, loading the data simultaneously, distribute the data across all disks. This eliminates hot spots (that is, areas where data might be concentrated).

5 ♦ After the data is loaded, perform a full index rebuild. Use the PROUTIL IDXBUILD utility.

13.8.6 Using No-integrity Mode Bulk Data Loading

PROUTIL can perform database updates faster when it runs in no-integrity mode rather than in the default full-integrity mode. However, if there is a system crash during this update activity, Progress cannot recover the database. The database can only be recovered by restoring a backup.

To enable no-integrity mode, use the No Crash Protection (`-i`) parameter. This parameter is particularly useful for running batch jobs, such as bulk data loading. For more information about No Crash Protection, see Chapter 18, "Database Startup Parameters."

CAUTION: Back up databases before you run any process with the `-i` startup parameter. Any crash that occurs while you are running processes with the `-i` parameter damages the database and makes that database invalid. If this happens, you must restore the database from the most recent backup copy.

13.8.7 Using 4GL Tools To Save Space By Deactivating Indexes

You can free up disk space for an application by deactivating its indexes. Also, if you are dumping and loading a large database, you can speed up the operation by deactivating the indexes in one or more database tables. This technique should be used only in situations where indexes are used infrequently, such as in an application that generates reports only once or twice a year. There are two ways to deactivate indexes:

- Deactivate individual indexes or all the indexes with the Data Administration tool if you are using a graphical interface, or the Data Dictionary if you are using a character interface.
- Deactivate indexes from a Progress 4GL procedure.

You can deactivate (but not reactivate) a single index from either Data tool. If you create a new unique index on an existing file, consider deactivating the index. If existing table data yields duplicate keys, all changes made during the current session are backed out when you try to save them. Create or reactivate a unique index after you have ensured that all key values are unique.

To activate an index, use the PROUTIL IDXBUILD utility. See Chapter 19, "Database Administration Utilities," for more information about activating indexes with the PROUTIL IDXBUILD utility.

Once an index is deactivated, you cannot use it to retrieve information from the records it normally points to. If you attempt to use a FOR, FOR EACH, or CAN-FIND statement in connection with a deactivated index, the database engine terminates the currently executing program. However, you can create, delete, or update a record that a deactivated index points to.

The database engine does not examine the active status of an index when it makes a search. Nor does the active status of an index affect the time stamp on the _Index file. As a result, precompiled programs do not require compilation if the only change to the schema is index activation or deactivation.

Deactivating Indexes

Follow these steps to deactivate an individual index or all of the indexes:

1 ♦ Access the appropriate Data tool (the Data Administration tool if you are using a graphical interface or the Data Dictionary if you are using a character interface).

2 ♦ Choose Utilities→ Deactivate Indexes. The Index Deactivation dialog box appears.

3 ♦ Choose OK. Progress lists the tables in the database.

4 ♦ Type **all** to deactivate all indexes, or select the indexes you want to deactivate. The Data tool prompts you to verify that you want to deactivate all the indexes.

5 ♦ Verify that you want to deactivate the specified indexes.

If you are using a graphical interface, follow these steps in the Data Dictionary to deactivate an individual index:

1 ♦ Access the Data Dictionary.

2 ♦ Choose the Index icon.

3 ♦ Choose the index you want to deactivate.

4 ♦ Choose the Index Properties button. The Index Properties dialog box appears.

5 ♦ Select the table that contains the index you want to deactivate. The Data Dictionary lists the indexes defined for the selected database.

6 ♦ Activate the Active toggle box.

7 ♦ Choose the Save button. The Data Dictionary prompts you to verify that you want to deactivate the index.

8 ♦ Verify that you want to deactivate the index. The Data Dictionary deactivates the index.

Deactivating Indexes With a Progress 4GL Procedure

You can also deactivate an index from a Progress 4GL procedure. Search through the _Index file until Progress finds the index you want to deactivate. Then set the _Active field equal to No. The following example uses this technique:

```
FIND _Index WHERE _Index-Name = "cust-num".
  IF (_Index._Active)
  THEN _Index._Active = NO.
  ELSE IF NOT(_Index._Active) THEN
  MESSAGE "The specified index is Deactivated.".
```

14

Managing Performance

The potential for improving performance depends on the type of system you run Progress on. Some options might not be available on your hardware or operating system platform. This chapter discusses options for managing database performance.

Specifically, this chapter contains the following sections:

- Introduction To Performance Managing
- Tools For Monitoring Performance
- Server Performance Factors
- CPU Utilization
- Disk I/O
- Memory Utilization
- Operating System Resources
- Database Fragmentation
- Index Use
- Virtual System Tables

14.1 Introduction To Performance Managing

The Progress Version 9 Database relies on the following system resources to perform its work:

- **CPU** — Manipulates data and executes programs
- **Disks and controllers** — Read and write data from database
- **Memory** — Stores data so it can be accessed quickly while performing operations
- **Operating system mechanisms** — Allocate and use system resources
- **Network** — Exchanges data between client and server systems

Performance is diminished if the database cannot use these resources efficiently. *Performance bottlenecks* occur when a resource performs inadequately (or is overloaded) and prevents other resources from accomplishing work. The key to improving performance is determining which resource is creating a bottleneck. Once you understand your resource limitation, you can take steps to eliminate bottlenecks.

Performance management is a continual process of measuring and adjusting resource use. Because system resources depend on each other in complex relationships, you might fix one problem only to create another. You should measure resource use regularly and adjust as required.

To effectively manage performance, you must have solid knowledge of your system, users, and applications. Because system and application performance can vary greatly depending on the configuration, use the information in this chapter as a guideline and make adjustments as required for your configuration.

The following additional factors that can affect performance:

- **System tuning** — Make sure your system is properly tuned and has sufficient capacity to perform its workload at the proper rate.
- **Other applications** — Make sure other applications are not competing with the database for system resources.

14.2 Tools For Monitoring Performance

This section describes several tools that you can use to monitor the performance of a Version 9 database. It includes information about:

- PROMON utility
- Virtual System Tables
- NT Performance Monitor

14.2.1 PROMON Utility

The Progress Monitor (PROMON) utility helps you monitor database activity and performance. Chapter 19, "Database Administration Utilities," documents the main PROMON options. In addition, PROMON provides advanced options (called R&D options) for in-depth monitoring of database activity and performance. The R&D options are described in Appendix A, "Progress Monitor R&D Options."

14.2.2 Virtual System Tables

Virtual system tables (VSTs) provide 4GL and SQL-92 applications access to the same database information that you can collect with the PROMON utility. The virtual system tables, or schema tables, have no physical records until the database manager generates them at run time. This enables a 4GL or SQL-92 application to retrieve information as run-time data.

14.2.3 NT Performance Monitor

The NT Performance Monitor is a graphical tool, supplied with Windows NT, that lets you monitor the performance of a local or remote Windows NT workstation. The Performance Monitor measures the performance of workstation objects like processes and memory using a defined set of counters. The Progress Version 9 database provides a comprehensive set of counters ranging from measurements about the number of clients running to the number of database records read or written. These counters are derived from the PROMON utility, Summary of Activity option, and they report on the state and performance of a particular database. Progress database support for the native NT performance monitor does not replace the PROMON utility; it simply provides another mechanism that system administrators can use to monitor performance-related data.

The Progress-specific counters are defined in the registry at installation. Each time you want to monitor database performance, you can specify the type of data that you want to monitor and how you want to monitor it. Note that your database must be registered through ProControl, otherwise, the NT Performance Monitor cannot use the Progress-specific counters.

Selecting Progress Counters

Follow these steps to select the counters whose performance you want to measure. Note that the counters you want to monitor must be active before you access the Performance Monitor:

1 ♦ Access the Performance Monitor by double-clicking its icon in the Administrative Tools program group.

2 ♦ Choose Edit→ Add to Chart.

3 ♦ From the Object selection list, choose Progress Version 9.

The Counter selection list then displays the Progress-specific counters.

4 ♦ Choose the counters that you want to measure by highlighting the counter name and choosing Add.

5 ♦ Choose Done when you have selected all of the counters that you want to measure.

To view a definition of the counter, highlight the counter name and choose Explain.

14.3 Server Performance Factors

When managing server performance, be aware of the following key factors:

- CPU utilization
- Disk I/O relating to database, before-image, and after-image I/O
- Record locking
- Memory utilization
- Database and index fragmentation

The following sections describe these factors.

14.4 CPU Utilization

To use your system to its full potential, the CPU should be busy most of the time.

Use operating system utilities to monitor CPU usage. When monitoring CPU usage, be aware of the following potential problems:

- Idle CPU
- Unproductive CPU processing

14.4.1 Idle CPU

If performance is inadequate and your CPU is idle, the CPU might be waiting for another resource. Identify the bottleneck and eliminate it so that the CPU can process work efficiently. Use PROMON to monitor database activity.

Disk I/O is a common bottleneck. For more information, see the "Disk I/O" section.

14.4.2 Unproductive CPU Processing

Symmetric multi-processing (SMP) systems use a spin lock mechanism to give processes exclusive access to data structures in shared memory. Spin locks ensure that only one process can use the structure at a time, but that all processes can get access to these structures quickly when they have to. However, if tuned incorrectly, SMP CPUs might spend time processing the spin locks unnecessarily instead of performing useful work.

The spin lock algorithm works as follows: When a process requires a shared-memory resource, it attempts to acquire the resource's latch. When a process acquires the latch, it has exclusive access to the resource. All other attempts to acquire the latch fail until the holding process gives up the latch. When another process requires access to the resource, it attempts to acquire the latch. If it cannot acquire the resource's latch because another process is holding it, the second process continues the attempt. This iterative process is called *spinning*. If a process fails to acquire a latch after a specified number of spins, the process pauses, or takes a *nap*, before trying again. If a process repeatedly fails to acquire a latch, the length of its nap is gradually increased. You can set the Spin Lock Tries (`-spin`) parameter to specify how many times to test a lock before napping.

To use a system of semaphores and queues to control locking, set `-spin` to zero (0).

Use the PROMON R&D "Adjust Latch Options" option under Administrative Functions to change the spin mechanism after start up.

14.5 Disk I/O

Because reading and writing data to disk is a relatively slow operation, disk I/O is a common database performance bottleneck. The Progress database engine performs three primary types of I/O operations:

- Database I/O
- Before-image I/O
- After-image I/O (relevant only if after-imaging is enabled)

If performance monitoring indicates that I/O resources are overloaded, try the techniques in the following sections to better balance disk I/O.

The best way to reduce disk I/O bottlenecks is to spread I/O across several physical disks, allowing multiple disk accesses to occur concurrently. You can extend files across many disk volumes or file systems.

NOTE: For a complete description of storage areas, see Chapter 1, "The Progress Database."

14.5.1 Database I/O

Database I/O occurs when the database engine reads and writes blocks containing records to and from disk into memory. To minimize database disk I/O, the database engine tries to keep a block in memory after it reads the block the first time. The next time the engine needs that block, it can access it from memory rather than reading it from disk.

To eliminate database I/O bottlenecks, you can:

- Increase the number of database buffers
- Change the number and structure of database storage areas
- Use private read-only buffers
- Use asynchronous page writers (APWs)

Storage Areas

Storage areas are the largest physical unit of a database. Storage areas consist of one or more extents that are either operating system files, operating system device raw partitions, or some other operating system level device that is addressed randomly. A storage area is a distinct address space, and any physical address stored inside the area is generally stored relative to the beginning of the storage area.

Storage areas give you physical control over the location of specific database objects. You can place each database object in its own storage area or place many database objects in a single storage area. Storage areas can contain database objects of one type or of many types. For example, to achieve load balancing, you can place a particularly active table in a separate storage area, then place the most active index for that table in its own storage area. Then, in a third storage area, place all the remaining tables and indexes. You cannot split a table or index across storage areas.

However, you can improve performance by moving tables and indexes to an application data storage area on a faster disk, while the database remains online. For a description of how to move tables and indexes while the database remains online, see Chapter 9, "Maintaining Database Structure."

Database Buffers

A *database buffer* is a temporary storage area in memory used to hold a copy of a database block. When the database engine reads a database record, it stores the block that contains that record in a database buffer. Database buffers are grouped in an area of memory called the *buffer pool*. Figure 14–1 illustrates database disk I/O.

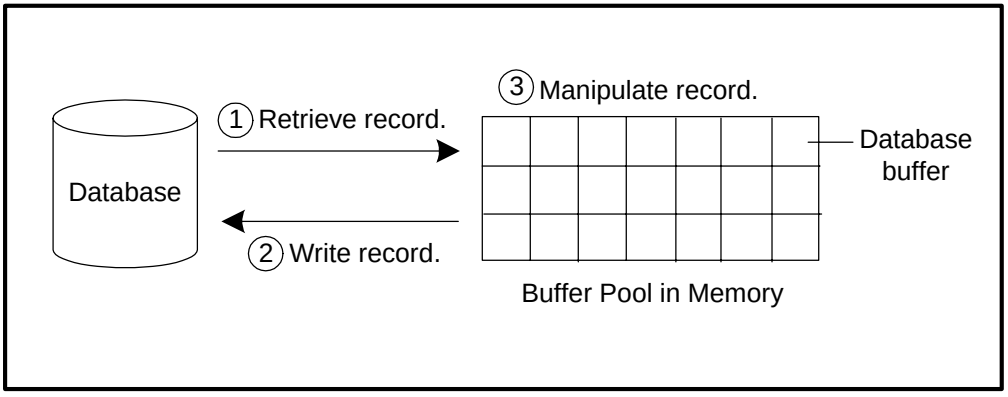


Figure 14–1: Database I/O

Progress database I/O works as follows:

1. When a process needs to read a database record, it requests access to the record.
2. The database engine searches the buffer pool for the requested record.
3. If the block that holds the record is already stored in a buffer, the engine reads the record from the buffer. This is called a *buffer hit*. When tuned correctly, the engine should achieve a buffer hit most of the time.

4. If the record is not found in any buffer, the engine must read the record from disk into a buffer. If an empty buffer is available, the engine reads the record into that buffer.

5. If no empty buffer is available, the engine must replace another buffer to make room for it.

   If the block that will be evicted has been modified, the engine must write the block to disk to save the changes. This is known as an *eviction*. While the eviction takes place, the process that requested the record in Step 1 must wait. For this reason, performance is improved if empty buffers are always available. See the "How the Database Engine Writes Modified Buffers" section for detailed steps.

Figure 14–2 illustrates how the engine reads a database record into a buffer.

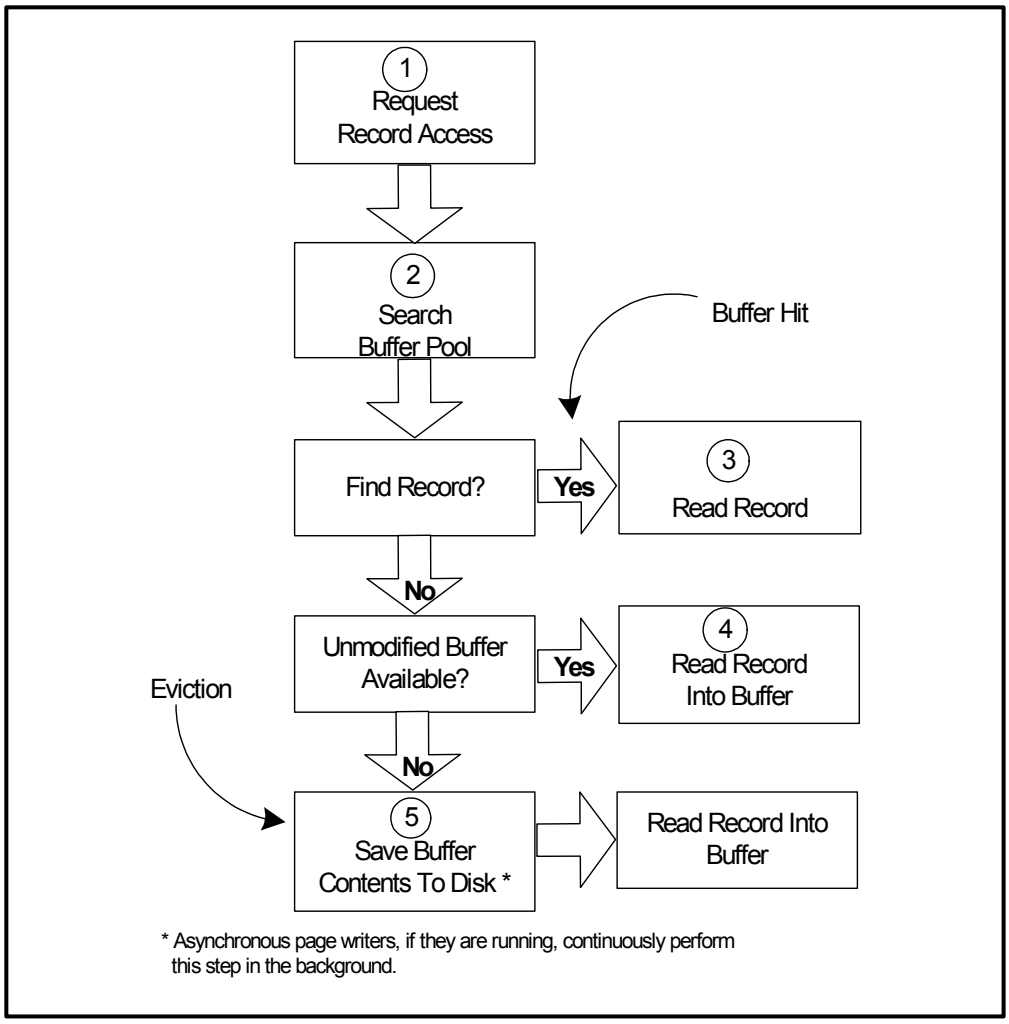


Figure 14–2: Reads and Writes To Database Buffers

How the Database Engine Writes Modified Buffers

When a process requires access to a database block that is not in the buffer pool, the database engine must replace another buffer to make room for it. The server searches for a buffer to replace.

The ideal replacement candidate is a buffer that is unlocked and unmodified. Replacing an unmodified buffer requires only one step: writing the new contents into the buffer. If a buffer contains modified data, it must first be evicted before it can be replaced. Evicting the buffer requires two steps: writing the buffer's contents to disk, then writing new contents into the buffer. It is therefore slower and requires more overhead. See Figure 14–3.

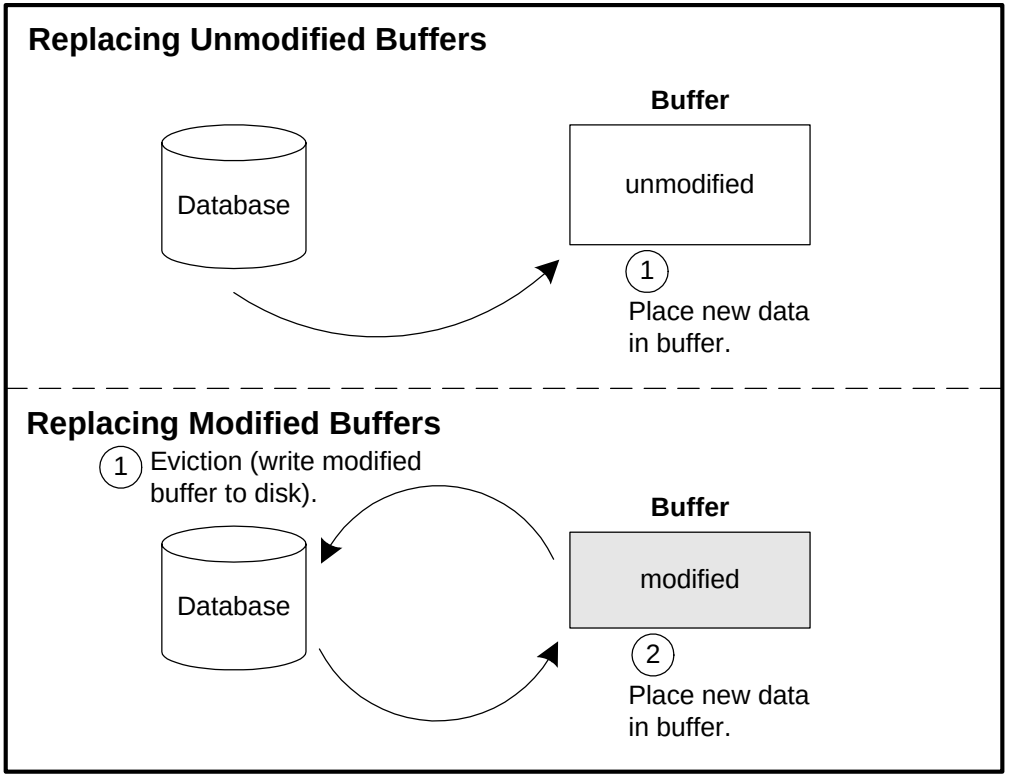


Figure 14–3: Evicting Buffers

When searching for a replacement candidate, the server searches a maximum of ten buffers. If the server fails to find an unlocked, unmodified buffer, the server evicts the first unlocked, modified buffer that it finds.

Monitoring Database Buffer Activity

A buffer hit occurs when the database engine locates a record in the buffer pool and does not have to read the record from disk. See the "Database Buffers" section for an explanation of buffer hits and how they improve performance by reducing overhead. When tuned correctly, the engine should achieve a buffer hit most of the time.

To determine the efficiency of database buffer activity, check the Buffer Hits field of the PROMON Activity option. For best performance, increase the Blocks in Database Buffers (-B) parameter until the buffer hits percentage exceeds 95 percent, or until your system starts paging. Figure 14–4 shows the Buffer Hits field in a sample Activity display.

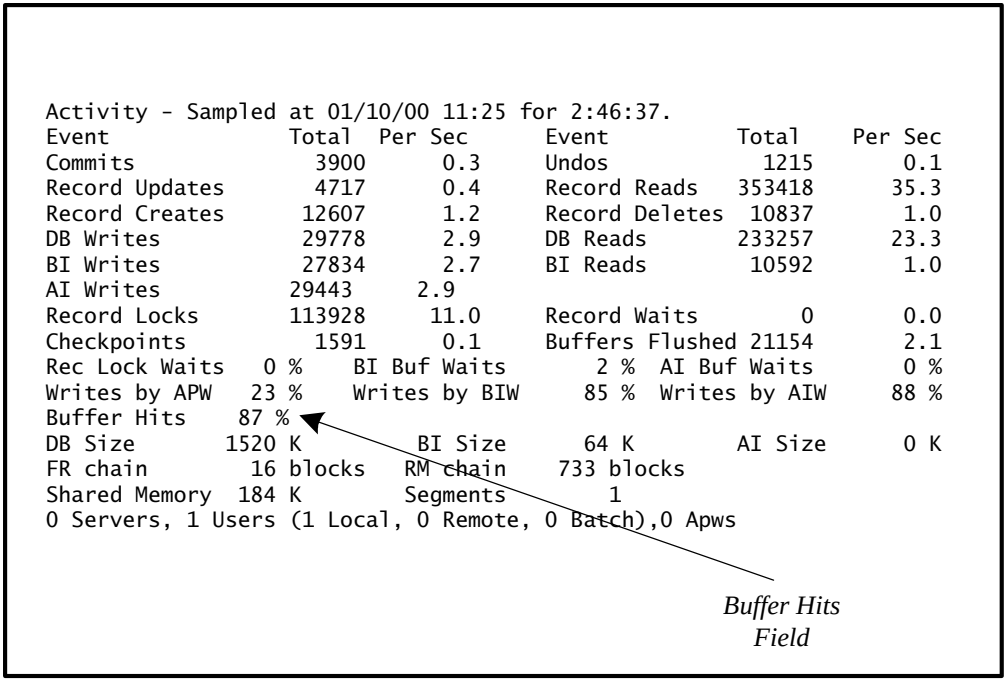


Figure 14–4: Monitoring Buffer Activity

Tuning Database Buffers

If database buffer monitoring indicates that buffer hits are below 85 percent, you can increase the number of database buffers so Progress can keep more records in memory.

To increase the number of buffer hits, increase the -B startup parameter to allocate more buffers. Increase the -B parameter until the buffer hits percentage exceeds 95 percent, or until your system starts paging.

The database engine uses a hash table to reduce the time it takes to locate a database buffer. The Hash Table Entries (-hash) startup parameter controls the number of hash table entries in the buffer pool. The database engine sets this parameter to approximately 25 percent of the number of database buffers (specified by the -B parameter). In most cases, the default value is adequate. However, increasing this parameter might slightly reduce the time required to find a block in the buffer pool.

Using Private Read-only Buffers

The buffer pool is a mechanism that conserves I/O when multiple users are accessing information from the same disk blocks. The buffer pool has a predefined size. Once the buffer pool becomes full, buffers are replaced on a least recently used (LRU) basis. Since sequential readers of the database access so many different buffers, they sometimes monopolize the buffer pool. That is, sequential readers of the database cause many shared buffers to be replaced by the buffers most recently used by the sequential reader.

Consequently, you can request some number of buffers in the buffer pool to be private read-only buffers. Private read-only buffers do not participate in the LRU replacement algorithm of the general shared buffer pool.

Applications that read many records in a short time, such as applications that generate reports or lists, should use private read-only buffers. Private read-only buffers prevent applications from quickly using all the public buffers and depriving buffers from other users. When an application is using private read-only buffers, updates are performed correctly using the public buffers. Therefore, an application performing many read operations but only a modest amount of updates might also benefit from using private read only buffers.

When a sequential reader is using private read-only buffers and needs a buffer to perform a read operation, and the buffer is already in the private read-only buffer pool, the database engine marks the buffer as most recently used (MRU) and uses it. If the buffer is not already in the private read-only buffer pool, the sequential reader takes a buffer from the LRU chain and puts it in the private read-only buffer pool. If the sequential reader has exhausted its quota of private read-only buffers, a private read-only buffer is replaced. The sequential reader maintains a list or chain of all its private buffers and uses a private LRU replacement mechanism identical to the public-shared buffer pool LRU replacement algorithm.

All users, regular and sequential, have access to all buffers in the buffer pool (public or private). If a regular user needs a block found in a private buffer pool, the buffer is removed from the sequential readers list of private buffers and is put back into the LRU chain as the most recently used buffer. In addition, if a sequential read user needs to update a private read-only buffer, it is removed from the sequential reader’s private buffer pool and put into the general shared buffer pool as most recently used.

Sequential reads use an index and require that index blocks be available in memory because they are used repeatedly. Therefore, you want to request enough private read-only buffers to hold all of the index blocks needed to retrieve a record. To determine how many private read-only buffers to set, count the number of tables that you read and determine the indexes you use. Then, determine the number of levels in the B-tree (balance tree) of each index and add 1 (for the record blocks). For example, request at least five private read-only buffers if you have a report that reads the Customer table using the Cust-Name index, and the Cust-Name index has four B-tree levels.

If you do not know the number of levels in your index, you can generally request six private read-only buffers and get a good result. If you perform a join and are reading from two tables simultaneously, request 12. If the system is unable to allocate the requested number of private read-only buffers, a message is written to the database log.

You can request a number of private read-only buffers using the Private Buffers (-Bp*n*) startup parameter. When you use the -Bp startup parameter the request remains active for the entire session unless it is changed or disabled by an application. Each user of private read-only buffers reduces the number of public buffers (-B).

NOTE: The total number of private read-only buffers for all simultaneous users is limited to 25% of the total blocks in database buffers. This value is set by the -B startup parameter. The maximum value of -B is 500,000. The single-user default is 10, and the multi-user default is 8*the number of users (specified by the maximum users (*n*) parameter).

You can also request a number of private read-only buffers from within a 4GL or SQL-92 application by setting a value in the _MyConn-NumSeqBuffers field of the _MyConnection virtual system table (VST). Since _MyConnection is an updatable virtual system table, private read-only buffers can be dynamically requested and released in the application. For a description of the _MyConnection VST, see Chapter 20, “Virtual System Tables.”

The following 4GL code example demonstrates how to turn private read-only buffers on and off:

```
/*Get 6 private read-only buffers for my application*/
FIND _MyConnection.
_MyConnection._MyConn-NumSeqBuffers = 6.

/**** Report using private read only buffers ***/
/* Turn off private read only buffers of my application */
FIND _MyConnection.
_MyConnection._MyConn-NumSeqBuffers = 0.
```

The following example demonstrates how to turn private read-only buffers on and off using an SQL-92 statement:

```
UPDATE pub."_MyConnection" SET "_MyConn-NumSeqBuffers" = 6.

UPDATE pub."_MyConnection" SET "_MyConn-NumSeqBuffers" = 0.
```

Using APWs To Improve Performance (Shared-memory Systems Only)

On shared-memory systems, you can use APWs to continually write modified database blocks to disk. APWs are optional, but highly recommended. They improve performance in the following ways:

- They ensure that a supply of empty buffers is available so the database engine does not have to wait for database buffers to be written to disk. For more information, see the "How APWs Provide a Supply Of Empty Buffers" section.
- They reduce the number of buffers that the engine must examine before writing a modified database buffer to disk. To keep the most active buffers in memory, the engine writes the least recently used buffers to disk; the engine must search buffers to determine which one is least recently used. For more information, see the "How APWs Reduce Buffer Replacement Search Times" section.
- They reduce overhead associated with checkpointing because fewer modified buffers have to be written to disk when a checkpoint occurs. For more information, see the "How APWs Reduce Checkpoint Overhead" section

You must manually start APWs. You can start and stop APWs at any time without shutting down the database. See Chapter 5, "Starting Up and Shutting Down," for instructions on starting and stopping an APW.

A database can have zero, one, or more APWs running simultaneously. The optimal number is highly dependent on your application and environment. To start, use one page writer for each disk where the database resides. Monitor the buffers flushed at checkpoint to determine if you need more.

NOTE: If you perform no updates, no page writers are required.

APWs are self-tuning. Once you determine how many APWs to run, you do not have to adjust any startup parameters specific to APWs. However, you might want to increase the BI cluster size to allow them to perform at an optimal rate. The PROUTIL TRUNCATE BI qualifier lets you create a BI cluster of a specific size. For more information, see Chapter 19, "Database Administration Utilities."

How APWs Provide a Supply Of Empty Buffers

APWs continually write modified buffers to disk, making it more likely the server will find an unmodified buffer without having to wait. To find modified buffers, an APW scans the Block Table (BKTBL) chain. The BKTBL chain is a linked list of BKTBL structures, each associated with a database buffer. Each BKTBL structure contains a flag indicating whether the associated buffer is modified. When an APW finds a modified buffer, it immediately writes the buffer to disk. Figure 14–5 illustrates how an APW scans the BLKTBL chain.

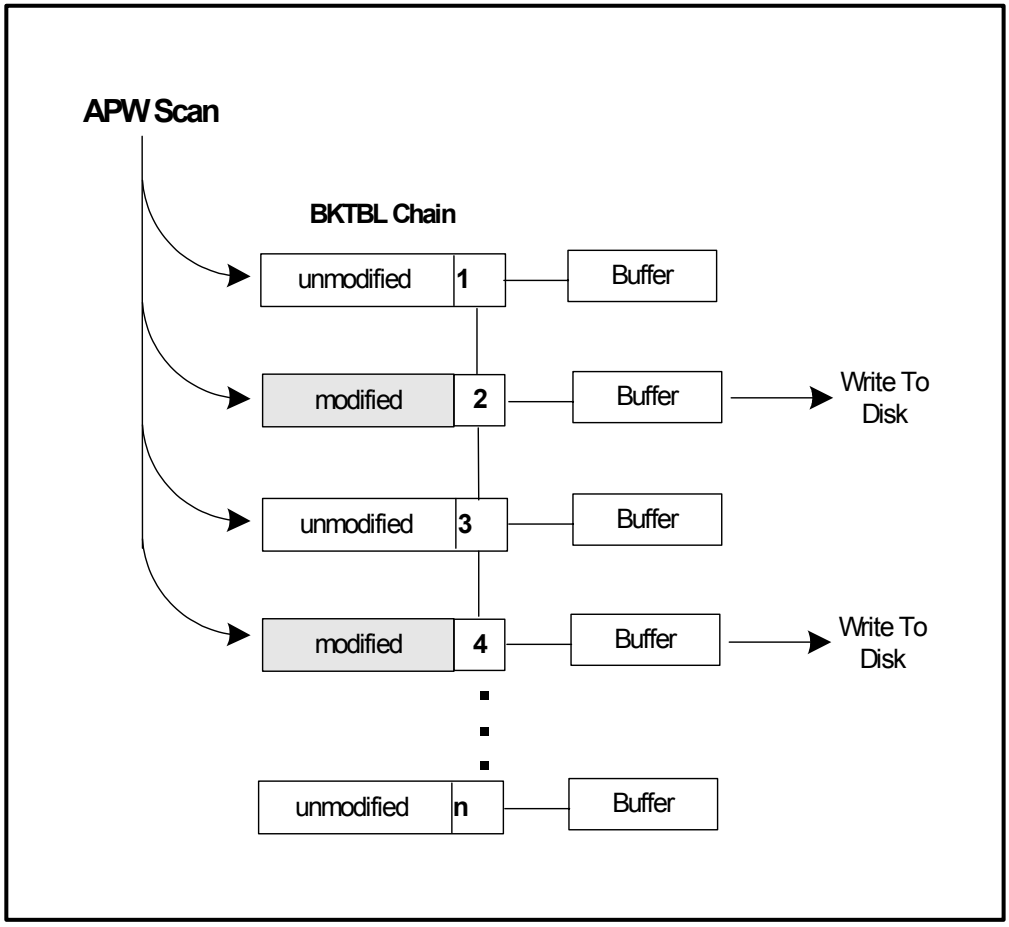


Figure 14–5: Block Table (BLKTBL) Chain

The APW scans in cycles. After completing a cycle, the APW goes to sleep. When the APW begins its next scanning cycle, it picks up where it left off. For example, if the APW scanned buffers 1 to 10 during its first cycle, it would start at buffer 11 to begin its next cycle.

How APWs Reduce Buffer Replacement Search Times

When the database engine writes modified buffers to disk, it replaces the buffers in a least-to-most-recently-used order. This is beneficial because you are less likely to need older data.

To find least recently used buffers, an APW scans the least recently used (LRU) chain. The *least recently used chain* is a doubly linked list in shared memory that the engine uses to access database buffers. The LRU chain is anchored by a data structure that points to the head and tail of the chain. Whenever a process accesses a database buffer, the server must lock and update the LRU anchor, moving the accessed buffer to the tail of the chain. Figure 14–6 illustrates the LRU chain.

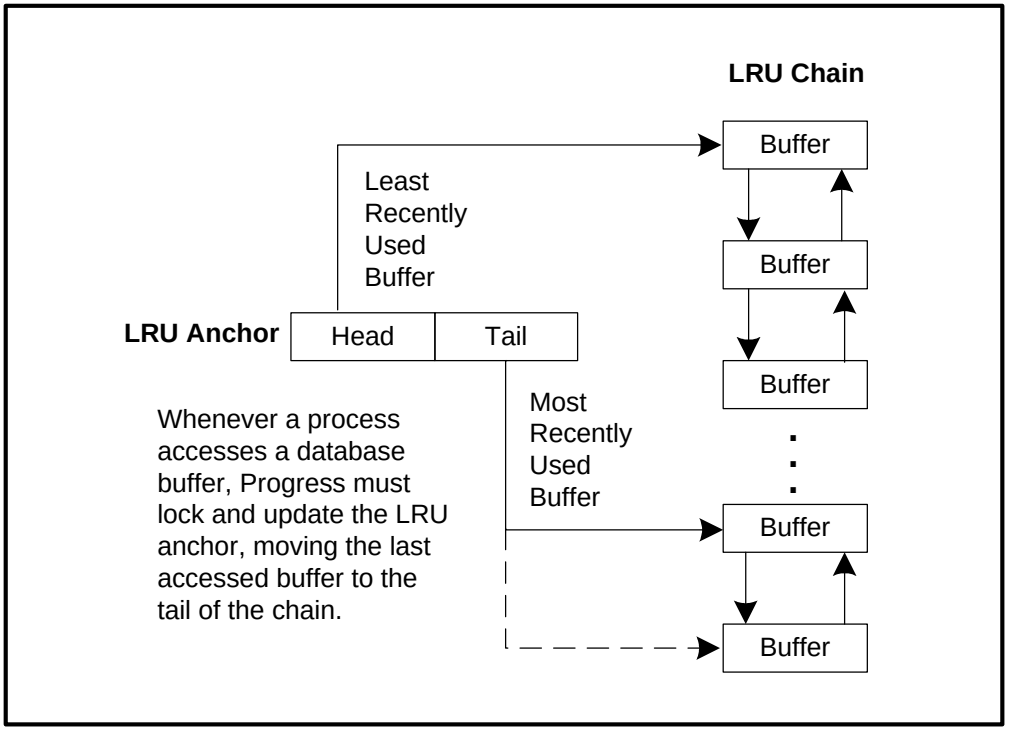


Figure 14–6: APWs and the Least Recently Used Chain

Since all processes must lock the LRU anchor whenever they have to access a buffer, long buffer replacement searches create contention for all processes accessing the database buffer pool. This can have a debilitating effect on performance, especially on heavily loaded systems. APWs reduce contention for the LRU anchor by periodically clearing out modified buffers. When buffer replacement is required, the database engine can find an unmodified buffer quickly.

How APWs Reduce Checkpoint Overhead

A third way that APWs improve performance is by minimizing the overhead associated with before-image checkpointing.

The before-image (BI) file is divided into clusters. A checkpoint occurs when a BI cluster becomes full. When a cluster becomes full, the database engine reuses the cluster if the information stored in it is no longer required. By reusing clusters, the engine minimizes the amount of disk space required for the BI file.

Checkpoints ensure that clusters can be reused and that the database can be recovered in a reasonable amount of time. During a checkpoint, the engine writes all modified database buffers associated with the current cluster to disk. This is a substantial overhead, especially if you have large BI clusters and a large buffer pool. APWs minimize this overhead by periodically writing modified buffers to disk. When a checkpoint occurs, fewer buffers must be written.

Monitoring APWs

The PROMON R&D option Page Writers Activity display shows statistics about APWs running on your system. Figure 14–7 shows a sample display.

```
01/25/00      Activity: Page Writers
16:29         from 01/25/00 13:56 to 01/26/00 11:23 (21 hrs 27 min)

                              Total   Per Min   Per Sec     Per Tx
Total DB writes               3       0         0.00        0.00
APW DB writes                 0       0         0.00        0.00
    scan writes               0       0         0.00        0.00
    APW queue writes          0       0         0.00        0.00
    ckp queue writes          0       0         0.00        0.00
    scan cycles               0       0         0.00        0.00
    buffers scanned           0       0         0.00        0.00
    bfs checkpointed          173     0         0.11        0.14
Checkpoints                   82110   0         5.22        6.79
Marked at checkpoint          0       0         0.00        0.00
Flushed at checkpoint         0       0         0.00        0.00

Number of APWs:                          1
```

Figure 14–7: PROMON Page Writers Activity Display

NOTE: Non-zero numbers in the “Flushed at Checkpoint” row indicates that the APW was unable to write buffers fast enough to prevent a memory flush. Increase the number of APWs and/or increase the cluster size to eliminate the flush.

14.5.2 Before-image I/O

Before-imaging is always enabled to let the database engine recover transactions if the system fails. This mechanism is extremely important for database reliability, but it creates a significant amount of I/O that can affect performance. In addition, before-image I/O is usually the first and most likely cause of I/O bottlenecks. The engine must always record a change in the BI file before it can record a change in the database and after-image (AI) files. If BI activity creates an I/O bottleneck, all other database activities are affected.

You can reduce the I/O impact of before-imaging by:

- Moving the BI file to its own disk
- Running a before-image writer (BIW) (shared-memory systems only)
- Providing more BI buffers
- Increasing the BI cluster size
- Increasing the BI block size
- Delaying BI writes

Monitoring BI Activity

Use operating system utilities to monitor the amount of I/O activity on the disk where the BI files reside. Use the PROMON utility to monitor specific BI activity. Use the R&D option BI Log Activity. Figure 14–8 shows a sample display.

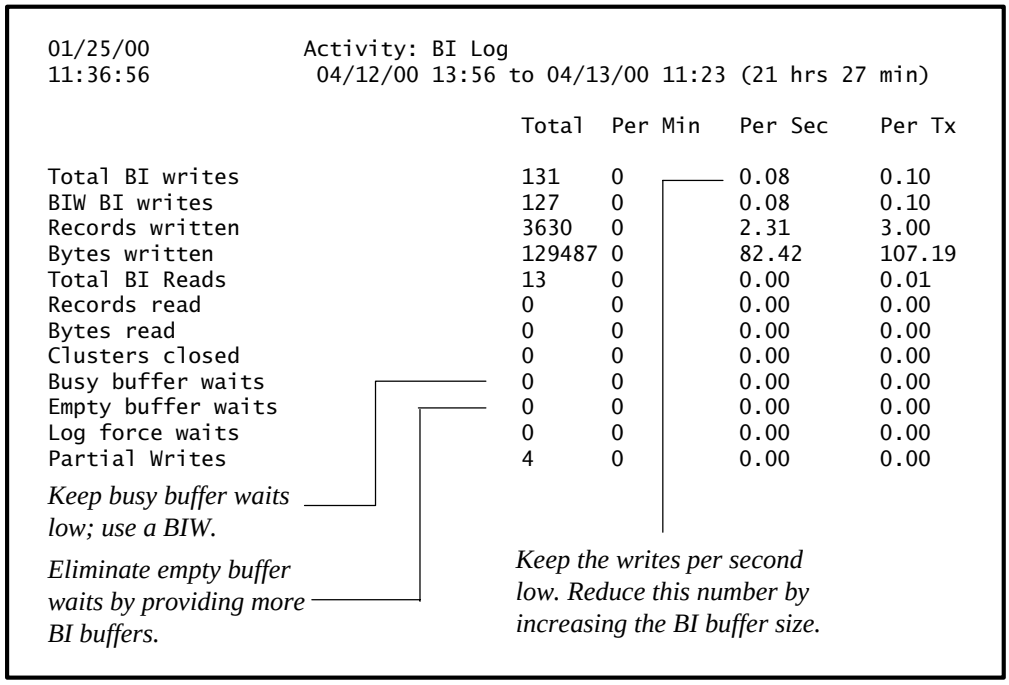


Figure 14–8: PROMON BI Log Activity Display

Look for the following potential problems:

- Busy buffer waits.
- Empty buffer waits.
- High number of writes per second.
- High number of partial writes. A *partial write* occurs when the database engine must write data to the BI file before the BI buffer is full. This might happen if:
  - An APW attempts to write a DB block whose changes are recorded in a BI buffer that has not been written. Because BI notes must be flushed before the AI note is flushed, the APW writes the data in the BI buffer before the buffer is full so it can perform the AI write.

- An after-image writer (AIW) runs ahead of the BIW. Because BI notes must be flushed before the AI notes can be written, the AIW writes the BI buffer before it is full so it can perform the AI write.
- The Suppress BI File Write (-Mf) parameter's timer expires before the buffer is filled.

Moving the BI File

The "Disk I/O" section explains the performance benefits of distributing database files across multiple disks. You help balance the before-image I/O load by placing the BI extents on a separate disk.

Using a Before-image Writer

The BIW is a background process that continually writes filled BI buffers to disk. Since writes to the BI file occur in the background, client and server processes rarely have to wait for a filled buffer to be written to disk. BIWs are optional, but highly recommended for improving I/O performance.

The server writes current information to the BI file through the current output buffer. When this buffer fills, the server places the buffer on the filled chain. The server then takes a new buffer from the empty chain and uses it as the current output buffer. If no empty buffers are available, the process must wait while a filled buffer is written to disk.

The BIW writes the filled buffers to disk and places them on the empty chain. By clearing out the filled chain, the BIW ensures that a supply of empty buffers is available to client and server processes.

You can only run one BIW per database. You must manually start the BIW, but you can start and stop the BIW process at any time without shutting down the database. See Chapter 5, "Starting Up and Shutting Down," for instructions on starting and stopping a BIW.

Providing More BI Buffers

You can increase the number of before-image buffers in the before-image buffer pool with the Before-image Buffers (-bibufs) startup parameter. Increasing the number of buffers increases the availability of empty buffers to client and server processes. In general, initially set this parameter to 20. Increase it if there are any empty buffer waits in the PROMON Activity screen or in the R&D BI Log Activity screen.

Increasing the BI Cluster Size

The BI file is organized into clusters on disk. As the database engine writes data to the BI file, these clusters fill up. When a cluster fills, the engine must ensure that all modified database buffer blocks referenced by notes in that cluster are written to disk. This is known as a *checkpoint*. Checkpointing reduces recovery time and lets the engine reuse BI disk space. Raising the BI cluster size increases the interval between checkpoints.

Raising the BI cluster size can reduce the I/O overhead of writing modified database buffers to disk. It also lets you defer writes and collect more changes before writing a block; this lets you write multiple changes with the same write.

Larger cluster sizes generally increase performance. However, they also have significant drawbacks:

- Increased disk space usage for the BI file.
- Longer crash recovery periods.
- Longer checkpoint times. (Run APWs to eliminate this drawback.)

Follow these steps to change the cluster size:

1 ♦ Use the PROSHUT command or the PROMON Shutdown a Database option to shut down the database.

2 ♦ Enter the following command:

```
proutil db-name -C truncate bi -bi size
```

For *size*, specify the new cluster size in kilobytes. The number must be a multiple of 16 in the range 16 to 262128 (16K–256MB). The default cluster size is 512K. Cluster sizes from 512 to 16384 are common.

You can also change the BI block size with this command. You might want to do so at this time. For more information, see the "Increasing the BI Block Size" section.

Increasing the Number Of BI Clusters

When you create a new database or truncate an existing database, the database engine, by default, creates four BI clusters, each of which is 512K. As the engine fills a cluster, the cluster is checkpointed, and the engine writes to the next cluster on the chain. Figure 14–9 illustrates the default BI clusters.

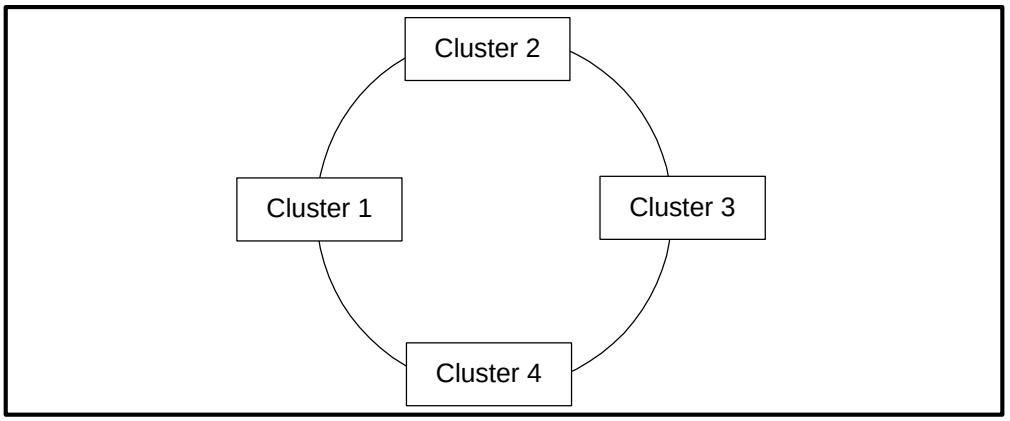


Figure 14–9: BI Clusters At Startup

In some cases, the database engine cannot write to the next cluster because the next cluster contains an active transaction. When the engine cannot use the next cluster on the chain, it creates a new cluster and begins writing to it. While the engine creates the new cluster, no database update activity can occur, thus impacting database performance. Figure 14–10 illustrates how BI clusters fill over time.

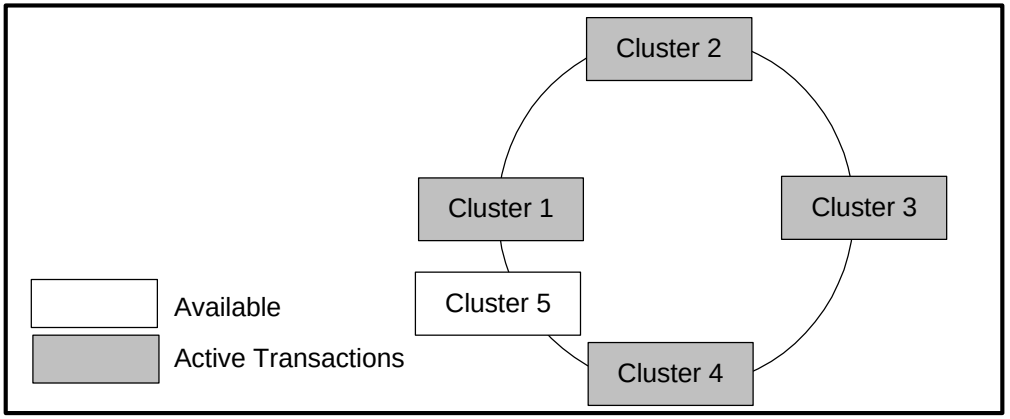


Figure 14–10: BI Clusters Over Time

The BI clusters typically grow to their optimal number over time. You can calculate the current number of BI clusters for a database by dividing the BI physical file size by the BI cluster size. For example, a database BI file with a BI cluster size of 128K and a physical size of 91,7504 has 7 BI clusters.

Whenever the BI file is truncated, you should consider growing the number of BI clusters to its optimal size before restarting the database, thus preventing the database engine from adding clusters on an as-needed basis. The BI file is truncated:

- Automatically by the database engine when you start after-imaging (RFUTIL AIMAGE BEGIN)
- Automatically by the database engine when you perform an index rebuild (PROUTIL IDXBUILD)
- Manually (PROUTIL TRUNCATE BI)

Follow this step to increase the number of BI clusters:

1 ♦ Enter the following command:

```
proutil db-name -C bigrow n
```

For *n*, specify the number of BI clusters that you want to create for the specified database. Progress creates four BI clusters by default. If you specify a BIGROW value of 9, Progress creates an additional 9 BI file clusters for a total of 13 clusters.

Increasing the BI Block Size

The database engine reads and writes information to the BI file in *blocks*. Increasing the size of these blocks allows the engine to read and write more data at one time. This can reduce I/O rates on disks where the BI files are located.

The default BI block size (8K) is sufficient for applications with low transaction rates. However, if performance monitoring indicates that BI writes are a performance bottleneck and your platform's I/O subsystem can take advantage of larger writes, increasing the BI block size might improve performance.

Follow these steps to change the BI block size:

1 ♦ Use the PROSHUT command or the PROMON Shutdown a Database option to shut down the database.

2 ♦ Enter the following command to change the BI block size:

```
proutil db-name -C truncate bi -biblocksize size
```

For *size*, specify the new BI block size in kilobytes. Valid values are 0, 1, 2, 4, 8, and 16. If you have a single AI file and after-imaging is enabled when you enter this command, you must use the After-image Filename (-a) parameter to specify the AI filename.

You can also change the BI cluster size with this command. You might want to do so at this time. For more information, see the "Increasing the BI Cluster Size" section.

For detailed information on this command, see the description of the PROUTIL utility in Chapter 19, "Database Administration Utilities."

Typically, if you change the BI cluster size and block size, you should also change the AI block size. For information, see the "Increasing the AI Block Size" section.

Delaying BI Writes

When the Delayed BI File Write (-Mf) startup parameter is set to zero, use the Group Commit technique to increase performance. This technique assumes that for the benefit of overall performance, each individual transaction can take slightly longer. For example, when a transaction begins to commit and spools its end note to the BI buffer, it waits a short time until one of two things happen: it fills the buffer and is written to disk, or a few other transactions complete and store their end notes in the BI buffer so that a single synchronous write commits all the transactions. Use the Group Delay (-groupdelay) startup parameter to set the amount of time (milliseconds) the transaction waits.

If the Group Commit technique does not provide sufficient improvement, you can improve performance on a busy system by delaying BI file writes with the Delayed BI File Write (-Mf) startup parameter.

By default, the database engine writes the last BI block to disk at the end of each transaction. This write guarantees that the completed transaction is recorded permanently in the database. On a system with little update activity, this extra BI write is very important and adds no performance overhead. On a busy system, however, the BI write is less important (the BI block will be written to disk very soon anyway) and might incur a significant performance penalty.

Set the -Mf parameter to delay BI writes at transaction commit time. When -Mf is set to a positive value, the last BI record is guaranteed to be written to disk within the specified number of seconds. The record is written sooner if the user logs out or the system shuts down.

NOTE: Suppressing the last BI write does not reduce database integrity. However, if there is a system failure, the last few completed transactions can be lost (never actually written to the BI file).

For more detailed information on the -Mf parameter, see Chapter 18, "Database Startup Parameters."

Setting a BI Threshold

When an application performs large schema updates or large transactions, the BI clusters can grow in excess of 2GB. If a crash occurs during such an operation, the recovery process might require twice the amount of disk space as the BI log was using at the time of the crash. Often this space is not available, leaving the database in an unusable state.

Using the Recovery Log Threshold (-bithold) startup parameter sets the maximum size to which BI files can grow. Once the threshold is reached, the database performs an emergency shutdown. This mechanism ensures that there will be enough disk space to perform database recovery. All messages associated with the threshold are logged in the database log (.lg) file. These messages include:

- Value of the threshold
- Warning message if the threshold is set above 1000MB
- Warning message when recovery log files are extended
- Message that a database shutdown is occurring because the threshold has been reached

The recommended range is to set -bithold between 3% and 100% of the largest possible recovery log file size, rounded to the nearest cluster boundary. If the threshold is set above 1000MB, Progress issues a warning message to the display and the database log (.lg) file. The system will check the total amount of BI clusters in use each time a new cluster is marked as used. If the No Crash Protection (-i) is set, the recovery log threshold parameter is set to the default (none) and cannot be overridden.

Enabling Threshold Stall

Often a database administrator does not want the database to perform an emergency shutdown when the Recovery Log Threshold limit is reached. The Threshold Stall (`-bistall`) startup parameter quiets the database when the recovery log threshold is reached. Instead of an emergency shutdown, the database stalls forward processing until the database administrator intervenes. This provides the database administrator the options of shutting down the database, making more disk space available, and increasing the threshold amount. A message is added to the database log (.lg) file stating that the threshold stall is enabled.

Using PROQUIET To Adjust the BI Threshold

You can adjust the value of the threshold by providing a valid threshold value for the PROQUIET command. The value can be increased above the current value or reduced to a value of one cluster larger than the recovery log file size at the time the PROQUIET command is issued.

Follow these steps to adjust the BI threshold:

1 ♦ Use the PROQUIET command to enable a database quiet point:

```
proquiet db-name enable
```

db-name is the name of the database for which you want to adjust the BI threshold.

NOTE: For more information on, and complete syntax for, the PROQUIET command, see Chapter 17, “Startup and Shutdown Commands.”

During a database quiet processing point, all file write activity to the database is stopped. Any processes that attempt to start a transaction while the quiet point is enabled must wait until you disable the database quiet processing point.

2 ♦ Adjust the threshold size using the `-bithreshold` parameter:

```
proquiet db-name -bithreshold n
```

db-name

Specifies the name of the database for which you want to adjust the BI threshold.

n

Specifies the new value for the threshold.

3 ♦ Use the PROQUIET command to disable the quiet point:

```
proquiet db-name disable
```

For more information on, and the complete syntax for, PROQUIET, see Chapter 17, "Startup and Shutdown Commands."

14.5.3 After-image I/O

After-imaging is an optional recovery mechanism that lets you recover data and transactions if a disk fails. AI files must be kept on separate disks from the database and BI files, so after-imaging I/O activity does not contribute to I/O activity on the disks where BI and database files are stored. However, after-imaging creates a significant amount of I/O that can affect performance. You can reduce the I/O impact of after-imaging by:

- Using an after-image writer (shared-memory systems only)
- Raising the AI block size

The following sections describe these options.

Monitoring AI Activity

Use operating system utilities to monitor the amount of I/O activity on the disk where the AI files reside.

Use the Progress PROMON utility to monitor specific AI activity. Use the R&D option AI Log Activity. Figure 14–11 shows a sample display.

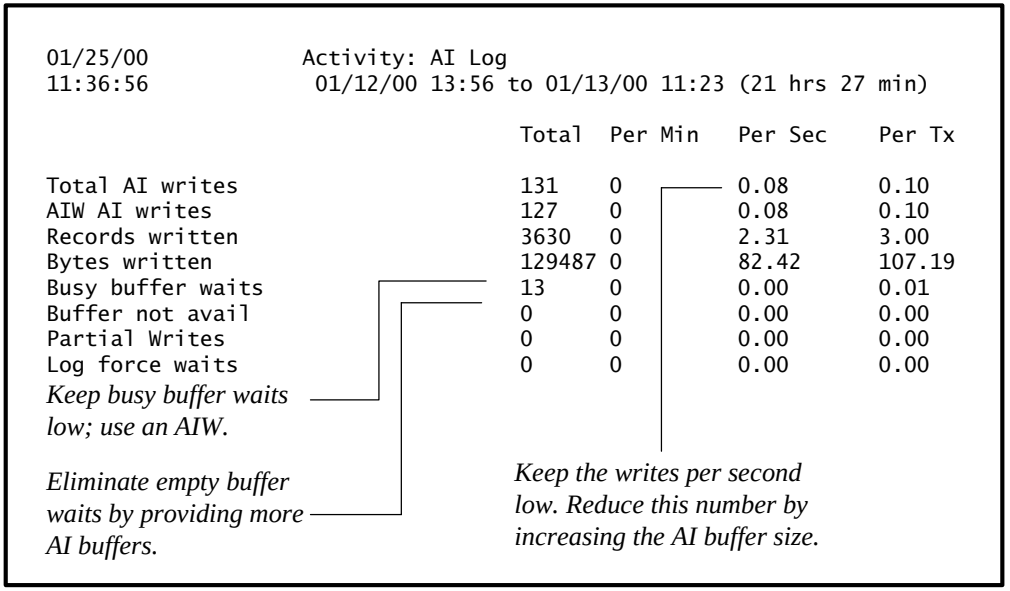


Figure 14–11: PROMON AI Log Activity Display

Using an After-image Writer (Shared-memory Systems Only)

The AIW is a background process that writes AI buffers to disk soon after they are filled. If the AIW is working effectively, client and server processes rarely have to wait for a modified buffer to be written to disk.

The AI buffer pool is a circular chain of buffers. The database engine fills these buffers one at a time. The buffer that the engine fills at any time is the current output buffer. As each buffer gets filled, the engine continues around the chain. Each buffer in turn becomes the current output buffer. If the next buffer to write is already modified, the engine must wait while that buffer is written to disk.

You can run only one AIW process per database at a time. You must manually start the AIW, but you can start and stop an AIW at any time without shutting down the database. See Chapter 5, "Starting Up and Shutting Down," for instructions on starting and stopping an AIW.

Increasing the `-aibufs` startup parameter increases the number of buffers in the after-image buffer pool, which increases the availability of empty buffers to client and server processes. Progress Software recommends you set the `-aibufs` parameter to 1.5 times the value of the Before-image Buffers (`-bibufs`) parameter. (For information on setting the `-bibufs` parameter, see the "Providing More BI Buffers" section.) Increasing `-aibufs` has no effect if the AIW is not running.

Increasing the AI Block Size

As with before-imaging, the database engine reads and writes information to the AI file in blocks. Increasing the size of AI blocks lets the engine read and write more AI data at one time. This can reduce I/O rates on disks where the AI files are located. In general, the default AI block size (8K) is sufficient for systems with low transaction rates. However, if performance monitoring indicates that AI writes are a performance bottleneck and your platform's I/O subsystem can take advantage of larger writes, increasing the AI block size might improve performance. A larger AI block size might also improve performance for roll-forward recovery processing.

Follow these steps to change the AI block size:

1 ♦ Use the PROSHUT command or the PROMON Shutdown a Database option to shut down the database.

2 ♦ If after-imaging is enabled, disable it by entering the following command:

```
rfutil db-name -C aimage end
```

For more specific information on this command, see the description of the RFUTIL utility AIMAGE END qualifier in Chapter 19, "Database Administration Utilities."

3 ♦ Truncate the BI file to bring the database and BI files up to date and eliminate any need for database recovery. To do this, enter the following command:

```
rfutil db-name -C truncate bi- [ -bi size | -biblocksize size ]
```

Typically, if you change the AI block size, you should also change the BI block size. If you have not already, you might want to use this command to do so. For more information on the BI block size, see the "Increasing the BI Block Size" section.

For more details on this command, see the description of the PROUTIL utility in Chapter 19, "Database Administration Utilities."

4 ♦ Change the AI block size by entering the following command:

```
rfutil db-name -C aimage truncate -aiblocksize size [ -a aifilename ]
```

For *size*, specify the size of the AI read and write block in kilobytes. The minimum value allowed is the size of the database block. Valid values are 0, 1, 2, 4, 8, and 16. If you specify 0, Progress uses the default size (8K) for your operating system platform.

5 ♦ Perform a full backup of the database.

NOTE: You must perform this step because backups and AI files created before you change the AI block size are no longer valid with the new AI block size. For detailed backup instructions, see Chapter 7, "Backing Up a Database."

6 ♦ Enable after-imaging by entering the following command:

```
rfutil db-name -C aimage begin { buffered | unbuffered } -a ai-name
```

For more specific information on this command, see Chapter 19, "Database Administration Utilities."

7 ♦ Restart the database and continue processing.

14.5.4 Direct I/O

The database engine can use an I/O technique that forces blocks to be written directly from the buffer pool to disk. This optional technique prevents writes to disk from being deferred by the operating system's buffer manager.

In general, use Direct I/O only if you are experiencing memory shortages. In many cases the normal buffered I/O will provide better performance. Test the performance impact before implementing Direct I/O on a production database.

To use this feature, specify the Direct I/O (`-directio`) startup parameter. If you use the `-directio` startup parameter, you might need to add additional APWs to compensate for the fact that with Direct I/O, each database write requires more processing time from the APWs.

14.6 Memory Utilization

Many of the techniques for improving server performance involve using memory to avoid disk I/O whenever possible. In general, you spend memory to improve performance. However, if the amount of memory on your system is limited, you can overload memory resources, causing your system to page. Paging can affect performance more severely than a reasonable amount of disk I/O. You must determine the point where memory use and disk I/O is balanced to provide optimal performance. In other words, you must budget the amount of memory you can afford to spend to improve performance.

14.6.1 Monitoring Memory Utilization

Use the following PROMON options to monitor memory usage:

- **Activity** — Shows the amount of shared memory used by the database and the number of shared-memory segments
- **Shared Resources Status (an R&D option)** — Shows the amount of shared memory allocated
- **Shared-memory Segments Status (an R&D option)** — Shows the ID number, size, and amount used for each shared-memory segment

For detailed information on these options, see Appendix A, “Progress Monitor R&D Options.”

14.6.2 Controlling Memory Use

Table 14–1 lists the startup parameters used to fine-tune memory allocation on the server system.

Table 14–1: Startup Parameters That Affect Memory Allocation

Startup Parameter	Suggested Use
Blocks in Database Buffers (-B)	Increasing the buffer size decreases the amount of database record I/O by increasing the number of buffers available in memory. This increases memory usage. Increase the -B parameter to use more memory to reduce I/O. Decrease the -B parameter if memory is limited or if database buffer I/O causes paging.
Maximum Clients per Server (-Ma)[1]	If some number of remote clients overloads the server or exhausts the file descriptors on a system, set this parameter to limit the number of clients.
Maximum Servers (-Mn)[1]	If a server becomes overloaded with clients, set this parameter to limit the number of servers. If you significantly increase this parameter, you should also increase the Minimum Clients per Server (-Mi) parameter.
Number of Users (-n)	On nonshared-memory systems, use this parameter to limit the total number of users below the level that overloads the database server. On shared-memory systems, set this parameter large enough to include both local and remote users.
Pin Shared Memory (-pinshm)	Use this parameter to prevent the database engine from swapping shared memory contents to disk.

[1] Relevant only to database accessed using client/server.

14.7 Operating System Resources

The database engine relies on operating system resources for its performance. For example, the engine uses the operating system file system and processes to perform its functions. These mechanisms are controlled by kernel parameters. The following sections describe these mechanisms and the kernel parameters associated with them.

14.7.1 Processes

On shared-memory systems, the following functions run as processes:

- Brokers
- Servers
- Clients
- APWs, BIWs, and AIWs

The user table contains one entry per process. Use the Number of Users (`-n`) parameter to specify the number of users.

On UNIX, the NPROC parameter limits the total number of active processes on the system and is commonly set between 50 and 200. The MAXUP parameter limits the number of concurrent processes that can be started by a single user ID, and it is commonly set between 15 and 25. Also, if more than one user logs in with the same user ID, MAXUP can be exceeded quickly. You see the following error message when the process limit is exceeded:

```
Unable to fork process.
```

If you see this message repeatedly, you should reconfigure your system kernel.

14.7.2 Semaphores

On single-processor systems, Progress uses semaphores to synchronize the activities of server and self-service client processes that are connected to a database. By default, each database has an array of semaphores, one for each user or server. Each process uses its semaphore when it must wait for a shared resource. Semaphores are not used for single-user sessions or for client sessions connecting to a remote database on a server system.

Figure 14–12 shows how semaphores control access to shared resources.

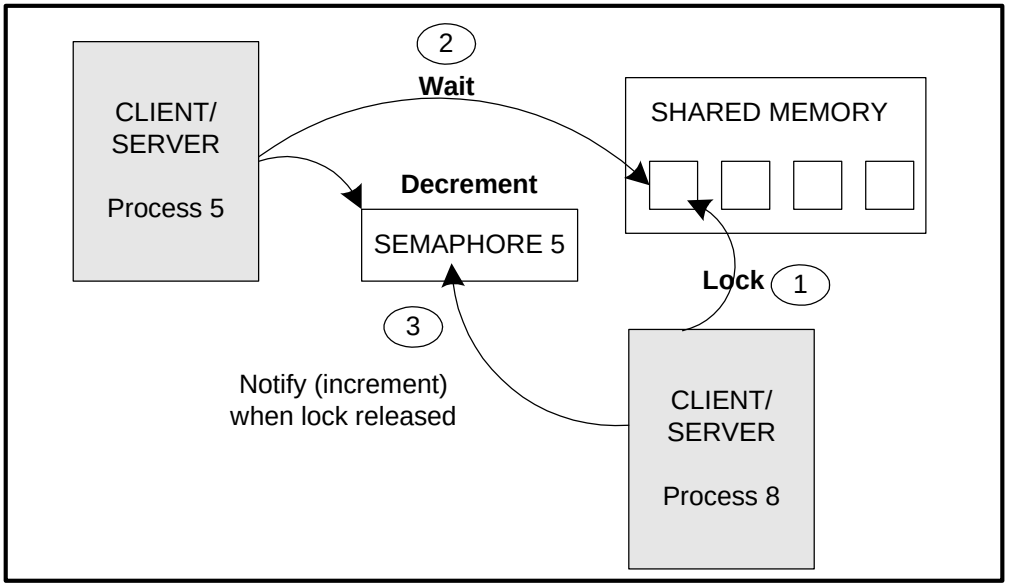


Figure 14–12: How a Semaphore Coordinates Concurrent Access

When process 5 needs access to a record, index, or other shared resource already locked by process 8, process 5 decrements its semaphore. When the process holding the lock (process 8) releases the resource, it notifies the waiting process (process 5) by incrementing the semaphore.

Semaphores are grouped into *semaphore sets*. Each semaphore set has a unique identifying number called a semid. Within a semaphore set, individual semaphores are identified by an integer ranging from 0 to one less than the size of the semaphore set.

The Progress broker preallocates semaphores when you start the database with PROSERVE. Each process requires one semaphore. The broker uses two additional semaphores internally. The database engine uses the following formula to determine the number of semaphores (#SEM) to allocate:

```
#SEM = Max-possible-users (-n) + Max-possible-servers (-Mn) + 4
```

Table 14–2 lists the UNIX kernel parameters that control the number and size of the semaphore sets.

Table 14–2: UNIX Kernel Parameters That Affect Semaphores

Parameter	Description	Recommended Setting
SEMMNI	The maximum number of semaphore identifiers allowed for the system.	One per active multi-user database. If you set this value too low, the database engine might generate error 1131.
SEMMSL	The maximum number of semaphores allowed per semaphore identifier (semaphore set).	(Max-local-users-on-any-databases + Max-#servers-on-any-databases + 4) If you set this value too low, the database engine might generate error 1093 or 1130.
SEMMNS	Total number of semaphores allowed for the system.	SEMMSL * number of active databases. If you set this value too low, the database engine might generate error 1093, 1131, or 1195.
SEMMNU	Maximum number of semaphore undo structures allowed for the system.	Same value as SEMMNS. If you set this value too low, the database engine might generate error 1081.

When you install Progress, you might have to increase the values of these parameters. If you are running other software that uses semaphores, take into account the combined requirements. See your system documentation for information on how to change these parameters.

The amount of kernel memory required for semaphores is relatively small, so setting the limits higher than your current needs probably will not affect performance.

The PROMON Shared Resources option (an R&D option) displays the number of semaphores used. When you start the broker process, a message specifies the number of semaphores still available. If the number of database users grows large, the database engine might exceed the maximum number of semaphores allowed, specified by the SEMMNS parameter. If this happens, you must reconfigure the system's kernel to increase the semaphore limit. You can reduce semaphore use only by lowering the values of the Number of Users (`-n`) and/or Maximum Servers (`-Mn`) startup parameters.

Allocating Semaphores

By default, the database engine uses one semaphore set for all the semaphores needed by the database. When greater than 1000 users connect to a single database, there might be high contention for the semaphore set. Using multiple semaphore sets helps alleviate this contention and improve performance with high user counts. The broker startup parameter, Semaphore Sets (`-semsets`), allows you to change the number of semaphore sets available to the Progress broker.

The broker uses two groups of semaphores, Login and User. The Login semaphore is used during connection to the database. The system allocates one User semaphore for every user specified by the Number of Users (`-n`) startup parameter. User semaphores are allocated using a round robin mechanism. If you specify the number of Semaphore Sets, Progress allocates one set for the Login semaphore and the remaining sets are used for User semaphores.

In this example, the broker uses two semaphore sets, one for the Login semaphore and one for the ten User semaphores:

```
proserve <db-name> -semsets 2 -n 10
```

In this example, the broker uses three semaphore sets, one for the Login semaphore, one for five of the User semaphores, and one for the remaining five User semaphores:

```
proserve <db-name> -semsets 3 -n 10
```

14.7.3 Spin Locks

On multi-processor shared-memory systems, the database engine uses a spin lock algorithm to control access to shared-memory structures. The spin lock algorithm works as follows: When a process needs a shared-memory resource, it attempts to acquire the resource's latch. If it cannot acquire the resource's latch it repeats the attempt. This iterative process is called spinning. If a process fails to acquire a latch after a specified number of spins, the process pauses, or takes a nap, before trying again. If a process repeatedly fails to acquire a latch, the length of its nap is gradually increased. You can set the Spin Lock Tries (`-spin`) parameter to specify how many times to test a lock before napping.

14.7.4 File Descriptors

A file descriptor is an identifier assigned to a file when it is opened. There is a system limit on the number of file descriptors. Each database process (clients, remote client servers, and the broker) might use 15 or more file descriptors. Therefore, set the system's file descriptor limit at approximately 15 times the maximum number of processes, or higher. Allow approximately10 file descriptors for the operating system as well.

14.8 Database Fragmentation

Over time, as records are deleted from a database and new records are added, gaps can occur on the disk where the data is stored. This *fragmentation* can cause inefficient disk space usage and poor performance with sequential reads. You can eliminate fragmentation by dumping and reloading the database.

14.8.1 Analyzing Database Fragmentation

To determine the degree of fragmentation for tables in a database, use PROUTIL with the TABANALYS qualifier:

```
proutil db-name -C tabanalys
```

You can run PROUTIL with the TABANALYS qualifier while the database is in use; however, PROUTIL generates only approximate information.

In the TABANALYS display, check the following fields:

- **Count** — The total number of record fragments found for each table in the database.
- **Fragments Factor** — The degree of record fragmentation for each table. If the value is 2.0 or greater, dumping and reloading will improve disk space usage and performance. If the value is less than 1.5, dumping and reloading is unnecessary.
- **Scatter Factor** — The degree of distance between records in the table. The optimal value for this field varies from database to database. To determine the optimal value for your database, run the TABANALYS qualifier on a freshly loaded database.

The following shows a sample excerpt from a PROUTIL TABANALYS display.

```
RECORD BLOCK SUMMARY FOR AREA "Info Area" : 7
-----------------------------------------------------
                                -Record Size (B)- ---Fragments--- Scatter
Table        Records      Size   Min   Max  Mean   Count  Factor     Factor
Customer          83     12.1K   118   222   149      83     1.0        1.5
Invoice          147     5613B    32    45    38     147     1.0        1.2
Item              55     5092B    39   229    92      55     1.0        1.3
Local-Default     10      704B    55    82    70      10     1.0        1.2
Ref-Call          13     2480B    81   328   190      13     1.0        1.4
Salesrep           9      746B    79    87    82       9     1.0        1.2
State             51     1755B    29    40    34      51     1.0        1.0


RECORD BLOCK SUMMARY FOR AREA "Order Area" : 8
-----------------------------------------------------
                                -Record Size (B)- ---Fragments--- Scatter
Table        Records      Size   Min   Max  Mean   Count  Factor     Factor
Order            207     11.5K    53    61    56     207     1.0        1.3
Order-Line       873     30.3K    33    38    35     873     1.0        1.2

          ------------------------------------------------------------------
Totals:         4190     431.1K     6  1156   105    4263     1.0        2.5
```

14.8.2 Eliminating Database Fragmentation

You can eliminate database fragmentation two ways:

1. Dump and reload the database.

   Dumping and reloading the database creates a new starting version of the database and loads the table contents into this new database. During the loading stage the Data tool (either the Data Dictionary or the Data Administration tool) or the 4GL procedure repartitions the disk, removing any gaps created when database elements were deleted, thereby more efficiently allocating disk space. For instructions, see Chapter 13, "Dumping and Loading."

2. Move individual tables and indexes.

   You can use the PROUTIL utility with the TABLEMOVE and IDXMOVE qualifiers to move tables and indexes from one storage area to another while the database remains online. As a result, disk space is efficiently allocated and gaps are removed. For more information about moving tables and indexes, see Chapter 9, "Maintaining Database Structure."

14.9 Index Use

As database blocks can become fragmented, index blocks can become under-utilized over time. The optimal degree of index block utilization depends on the type of database access performed. Retrieval-intensive applications generally perform better when the index blocks are close to full since the database engine has to access fewer blocks to retrieve a record. The larger the index, the greater the potential for improving performance by compacting the index. Update-intensive applications, on the other hand, perform better with loosely packed indexes because there is room to insert new keys without having to allocate more blocks. Index analysis provides the utilization information you require to make decisions. Choose a balance between tightly packed indexes and under-utilized indexes, depending on your data and applications, and use the Index Rebuild (IDXBUILD) qualifier, described later in this chapter, to compact indexes.

14.9.1 Analyzing Index Use

Use PROUTIL's IDXANALYS qualifier to get information about index blocks and utilization.

To execute the IDXANALYS qualifier, enter the following command:

```
proutil db-name -C idxanalys
```

db-name

Specifies the name of the database.

The IDXANALYS qualifier provides:

- The number of fields and levels in each index
- The size of each index, in blocks and in bytes
- The percent utilization within the index (that is, the degree of disk space efficiency)
- A factor value that indicates whether to rebuild each index
- A summary of indexes for the current database and the percentage of total index space used by each index

NOTE: You can run PROUTIL with the IDXANALYS qualifier while the database is in use; however, PROUTIL generates only approximate information.

The most important field in the IXANALYS display is the % Util field. This field shows the degree of consolidation of each index. If an index is several hundred blocks and your application most frequently retrieves data, an index utilization of 85 percent or higher is optimal. There are two ways to increase an index's utilization rate:

- Compress the index with the database online or offline with the PROUTIL IDXCOMPACT utility.
- Rebuild and compress the index offline with the PROUTIL IDXBUILD utility.

The Levels field shows the number of reads PROUTIL performs in each index per entry. The Blocks and Bytes fields show you the size of each index. The Factor field is based on the utilization and size of the index; it is an indicator of when you should rebuild indexes. Table 14–3 provides a description of the different ranges of values for the Factor field. When you use the Factor field to decide whether to rebuild an index, consider the context of how the particular index is used. For example, if an index is highly active, with continuous insertions and deletions, its utilization rate varies greatly, and a rebuild is inadvisable. However, a static index with a high factor value benefits from a rebuild.

Table 14–3: Factor Values

Factor Range	Description
1 to 2	The index is well-utilized and balanced. You do not have to rebuild it.
2 to 2.5	The index is less than 50 percent utilized and/or the index is unbalanced. You should consider a rebuild.
2.5 to 3	The index is less than 25 percent utilized and/or the index is very unbalanced. You should rebuild this index.

14.9.2 Compacting Indexes

When space utilization of an index is reduced to 60% or less as indicated by the PROUTIL IDXANALYS utility, use the PROUTIL IDXCOMPACT utility to perform index compaction online. Performing index compaction increases space utilization of the index block to the compacting percentage specified. For example:

```
proutil db-name -C idxcompact [ owner-name.]table-name.index-name [n]
```

NOTE: For the complete syntax description see Chapter 19, “Database Administration Utilities.”

Performing index compaction reduces the number of blocks in the B-tree and possibly the number of B-tree levels, which improves query performance.

The index compacting utility operates in phases:

- Phase 1: If the index is a unique index, the delete chain is scanned and the index blocks are cleaned up by removing deleted entries.
- Phase 2: The nonleaf levels of the B-tree are compacted starting at the root working toward the leaf level.
- Phase 3: The leaf level is compacted.

The _UserStatus virtual system table displays the utility's progress. For more information see Chapter 9, “Maintaining Database Structure.”

NOTE: Because index compacting is performed online, other users can use the index simultaneously for read or write operation with no restrictions. Index compacting only locks 1 to 3 index blocks at a time, for a short time. This allows full concurrency.

14.9.3 Rebuilding Indexes

Use the IDXBUILD (Index Rebuild) qualifier of the PROUTIL utility to:

- Compress index blocks to minimize space usage.
- Activate all deactivated indexes in the database.
- Repair corrupted indexes in the database. (Index corruption is normally signaled by error messages.)

NOTE: When you run the Index Rebuild qualifier, the database must not be in use.

To run the IDXBUILD qualifier with PROUTIL, enter the following command:

```
proutil db-name -C idxbuild [ all ] [ -TB ] [ -TM ]
```

db-name

Specifies the name of the database whose indexes you want to build.

To improve performance, use the Merge Number (-TM) and Speed Sort (-TB) startup parameters. For details, see the “Maximizing Index Rebuild Performance” section.

When you enter this command without the `all` qualifier, the following menu appears:

```
Index Rebuild Utility
=====================
Select one of the following:
All - Rebuild all the indexes
Some - Rebuild only some of the indexes
Quit - Quit, do not rebuild

Enter your selection:
```

Use the Some option to rebuild only specific indexes. Use the All option to rebuild all indexes. After you enter a selection and you name those indexes you want to rebuild, the utility prompts if you have enough disk space for index sorting. If you enter yes, the utility sorts the indexes you are rebuilding, generating the indexes in order by their keys. This sorting results in a faster index rebuild and better space use in the index blocks.

To estimate whether you have enough free space to sort the indexes or not, use the following formulas:

- If you rebuild all the indexes in your database, sorting the indexes requires up to 75 percent of the total database size in free space.
- If you rebuild an individual index, sorting that index requires as much as the following amount of free space:

  (*size of one index entry*) * (*number of records in file*) * 3

The Index Rebuild qualifier with PROUTIL rebuilds an index or set of indexes in a series of three phases:

1. The utility scans the database file, clearing all index blocks that belong to the indexes you are rebuilding and adding those blocks to the free block list.
2. The utility scans the database file and rebuilds all the index entries for every data record. If you chose to sort the index, the utility writes the index entries to the sort file. Otherwise, the utility writes the index entries to the appropriate index at this point.
3. This phase only occurs if you chose to sort the indexes. In this phase, the utility sorts the index entries in the sort file into groups and enters those entries into their respective entries in order, one index at a time.

The Index Rebuild qualifier accomplishes most of its work without displaying messages, unless it encounters an error condition.

If the index rebuild is interrupted while rebuilding selected indexes, the list of selected indexes is retained in a file named `dbname.xb`. This XB file is used when the utility is restarted. You do not have to enter the list of indexes manually if the XB file exists.

Overcoming SRT Size Limitations

When you perform the Index Rebuild utility and choose the Sort option, you might encounter space limitations that can cause the utility to terminate. These are some of the reasons that you might reach the limit:

- The temporary sort file allocated to do the sort reaches the operating system limit for file size (2GB on most systems).

  To overcome this limitation, simply create a file that contains specifications for the directories and the amount of space per directory that you want the SRT file to have access to during the Index Rebuild. The file that contains the specifications must be a text file, must have the same name as the database with an extension of .srt (*dbname*`.srt`), and must reside in the same directory as the .db file. In addition, the contents of the file must follow these conventions:

- List the directory and the amount of space that you want to allocate to the index rebuild sort on separate lines.

  The size that you specify in the *dbname*`.srt` directory specification is the maximum (in 1024 byte units) that the file can grow. Specifying 0 for any directory indicates that you want to allow unlimited growth.

  When you provide a file size that is larger than the maximum allowed by your operating system, Index Rebuild ignores the value you specified in the file and uses the maximum size allowed by the operating system. When you specify a nonzero file size that is less then the temporary block size (`-TB`), Index Rebuild ignores the value you specified in the file and uses the temporary block size (`-TB`) value.

- Separate the directories from the size by at least one blank.

- Terminate the line with a slash (/) followed by end of line.

For example, if you want to rebuild the index for the sports database and you want the speed sort to have access to 300K of space available in the `/user2/db1/first` directory, 400K in the `user3/junk` directory, and unlimited space in the /user4/last directory, then the `sports.srt` looks like this on UNIX:

```
300  /user2/db1/first/
400  /user3/junk/
0    /user4/last/
```

and looks like this for Windows:

```
300  \user2\db1\first\
400  \user3\junk\
0    \user4\last\
```

The Index Rebuild utility accesses the files in the order in which they are listed in the *dbname*`.srt` file. So, if you specify an amount of space that is not available, when the disk is filled, then Index Rebuild terminates and the next directory specification is not used. Thus, if a disk has only 200K of space and the `dbname.srt` specifies 300K, when the 200K is exhausted the Index Rebuild terminates. For example, if `/user2/db1/first` above does not get 300K of data, Index Rebuild never processes `/user3/junk`. In addition, if you specify a directory size of 0, any directories specified after it in the *dbname.srt* are not processed. For these reasons, you should verify that the space you specify in the *dbname.srt* file is available before running index rebuild.

The Index Rebuild utility opens the files for each of the directories before it actually starts the sort process. As a result, one of the following messages is displayed for each file:

```
Temporary sort file at pathname used up to nK of disk space.
```

or:

```
Temporary sort file at:pathname will use the available disk space.
```

The previous message occurs even if the .srt file was not found.

When the sort completes, the following message is displayed for each file:

```
Temporary sort file at pathname used nK of disk space.
```

In some cases the message displays OK. This simply means that the sort took place completely in memory.

If Index Rebuild does not find a *dbname*.srt file, then by default, it uses the directory supplied by either the -T parameter or the current working directory.

Maximizing Index Rebuild Performance

To speed up index rebuild operations, do the following:

- Answer yes when prompted whether you have enough disk space for sorting.
- Increase the Speed Sort (-TB) startup parameter to 24K. (If you are very short of memory, use 16K or 8K.) This improves sort performance; however, it also uses more memory and disk space. See the *Progress Startup Command and Parameter Reference* for more information about the -TB startup parameter.
- Increase the Merge Number (-TM) startup parameter to 32 (unless memory is scarce). The -TM parameter is described in the *Progress Startup Command and Parameter Reference*.
- Change the Temporary Directory (-T) startup parameter to store the temporary files on another disk. See the *Progress Startup Command and Parameter Reference* for more information on the -T parameter.

The database engine uses the following algorithm to rebuild indexes. For each record, read the index key fields and store in the first available SRT file block. Allocate additional SRT file blocks of the specified block size as required to hold all index keys. Sort the keys in each block then merge the keys to produce a sorted file. A similar technique is used to sort records when there is no index to satisfy a BY clause.

A larger block size can improve index rebuild performance considerably. A larger block size means less SRT block allocation overhead and fewer quicksort operations on individual blocks.

You might have to run the application several times using different block size values to determine the optimal value. If you experience extreme memory shortages when running Progress, try setting the block size to 1 to reduce memory consumption.

During index rebuild, try setting -TB to 31, if memory and disk space are available. If the index rebuild fails, try successively smaller values. Remember, a larger value for -TB improves sort performance but uses more memory. The -TB setting has a significant impact on the size of the SRT temporary file. The SRT file size depends on the number of session compile files, and the number and size of sort operations. (See the description of the Directory Size (-D) startup parameter in Chapter 18, “Database Startup Parameters.”

Memory usage depends on the number of nested FOR EACH statements doing sorts (number of simultaneous sorts occurring). You can estimate memory usage as follows, where M is the estimated memory usage:

M=(*sort-block-size*)*(*number-of-nested-sorts+Merge-Number(-TM)parameter*)

Index rebuild always requires eight simultaneous sorts, so during index rebuild:

M=(*sort-block-size*)*(8+(*-TM) parameter*)

Therefore, in the default case:

M=(2*(8+5))=26K

Reactivating Unique Indexes

When reactivating a unique index, IDXBUILD displays the following error message each time it encounters a duplicate index key:

```
Fix RECID recid, table-name already exists with field-name value.
```

You must change the record data to eliminate duplicate keys. Use another index on the table (if one exists):

```
FOR EACH table-name USE-NDEX index-without-duplicate-keys:
UPDATE table-name.
```

14.10 Virtual System Tables

Virtual system tables (VSTs) provide 4GL and SQL-92 access to system statistics. You can use this information to help analyze database and application design, as well as to determine what factors are impacting database performance. The virtual system tables, or schema tables, have no physical records until the database manager generates them at run time. For a detailed description of each table, see Chapter 20, “Virtual System Tables.”

15

Replicating Data

Data replication is the distribution of copies of information to one or more sites. In a single enterprise, sites spanning organizational and regional boundaries often require the timely sharing of transaction data across databases in a consistent manner. Developing and deploying a successful replication process involves careful planning and input from business experts, application developers, and administrators.

This chapter contains the following sections:

- Replication Schemes
- Implementing Log-based Site Replication

15.1 Replication Schemes

A replication scheme is a system of definable tasks and operations that are used to perform data replication. An enterprise's replication scheme addresses it's specific business requirements This section summarizes different replication models, data ownership models, and implementation strategies that are available. A replication scheme can be implemented through event triggers or through log-based capture.

15.1.1 Trigger-based Replication

To implement trigger-based replication, use event triggers stored in the database. When an event to be replicated occurs (that is, a record is created, modified, or deleted) the database uses the event to record the change in a replication change log. Progress 4GL provides full support for trigger-based replication. See the *Progress Programming Handbook* for more information about trigger-based replication.

15.1.2 Log-based Site Replication

Log-based site implementation (or site replication) is based on a monitor that watches a log of transactions and propagates these changes to other sites. Generated by the database, this log comprises a stream of database transactions. This is a very efficient way of replicating data from server to server. It also allows you to maintain a remote backup of a primary database. Progress supports site replication using after-imaging (AI) files. For more information on log-based site replication see "Implementing Log-based Site Replication."

15.2 Replication Models

At the highest level, there are two major models of replication: *synchronous* and *asynchronous*.

15.2.1 Synchronous Model

In a synchronous replication model, all replication of data occurs within the scope of the original transaction. In other words, replication occurs transaction by transaction. Typically, this model is implemented using a *two-phase commit* protocol. Two-phase commit ensures that distributed transactions occur consistently across databases. For more information, see Chapter 12, "Using Two-phase Commit."

Because the data modifications are replicated as part of the original transaction, synchronous replication ensures high data availability and consistency. The entire transaction is either committed to both systems or backed out completely.

15.2.2 Asynchronous Model

Asynchronous replication (also known as store and forward replication) allows the replication to occur outside the scope of the original transaction. The replication might take place seconds, minutes, hours, or days from the time of the transaction, depending on your business requirements. Although the replication executes record by record, replication can occur by transaction. That is, if an order is placed in the system with order lines containing multiple data changes and these changes are made within the scope of a single transaction, the changes can be replicated as a single transaction.

15.2.3 Database Ownership Models

Data ownership models determine how changes to a database (or site) affect other databases in the network. This section describes three models, *data distribution*, *data consolidation*, and the *peer-to-peer* model, and how they relate to replication.

Distribution Model

In the *distribution* ownership model, a single master database owns the data. The master database is the read/write area, and all changes are made to this database only. All changes are then propagated to the remote sites in a read-only state. The remote sites cannot change the data, only view it. In terms of replication, the chief advantage to this model is that it greatly reduces data collision (conflicts between updates to the same record). This is because data changes are made at one site only.

Figure 15–1 illustrates the data distribution model.

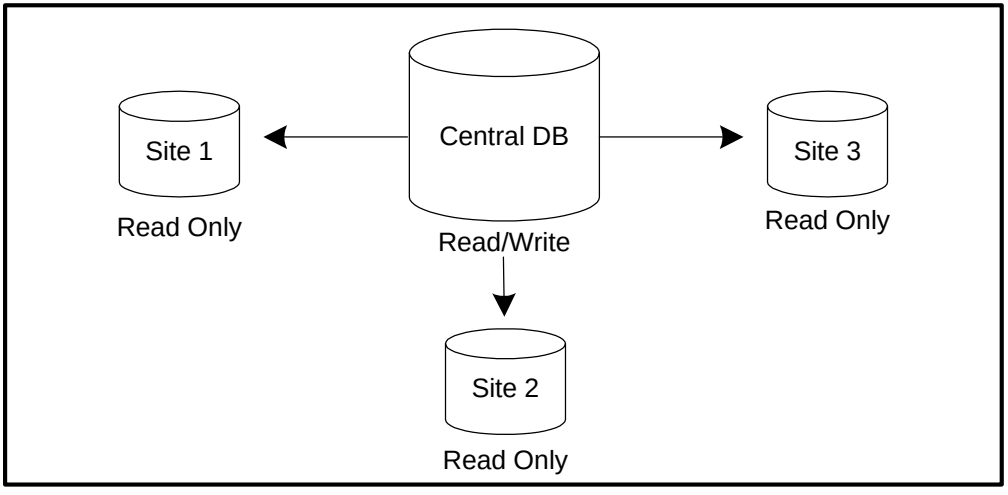


Figure 15–1: Data Distribution Model

Consolidation Model

In the *consolidation* model, data changes are made at the remote sites and then propagated to the central database. The central database is read-only and is used for reporting purposes. For replication, this model increases the frequency of data collision over the distribution model. If there is a collision of changes by two or more users, the changes are applied on a first-come-first-served basis.

To avoid data collision, the consolidation model often uses *table partitioning.* Table partitioning (also called data ownership) requires that all data be owned by each site. Changes to data at each remote site are made exclusively by respective remote site users. A data ownership model might not be appropriate for your business organization. Although data collisions are avoided, the ability to update the same record from any site is lost.

Figure 15–2 illustrates two data consolidation models, one with no data ownership, and the other with table partitioning.

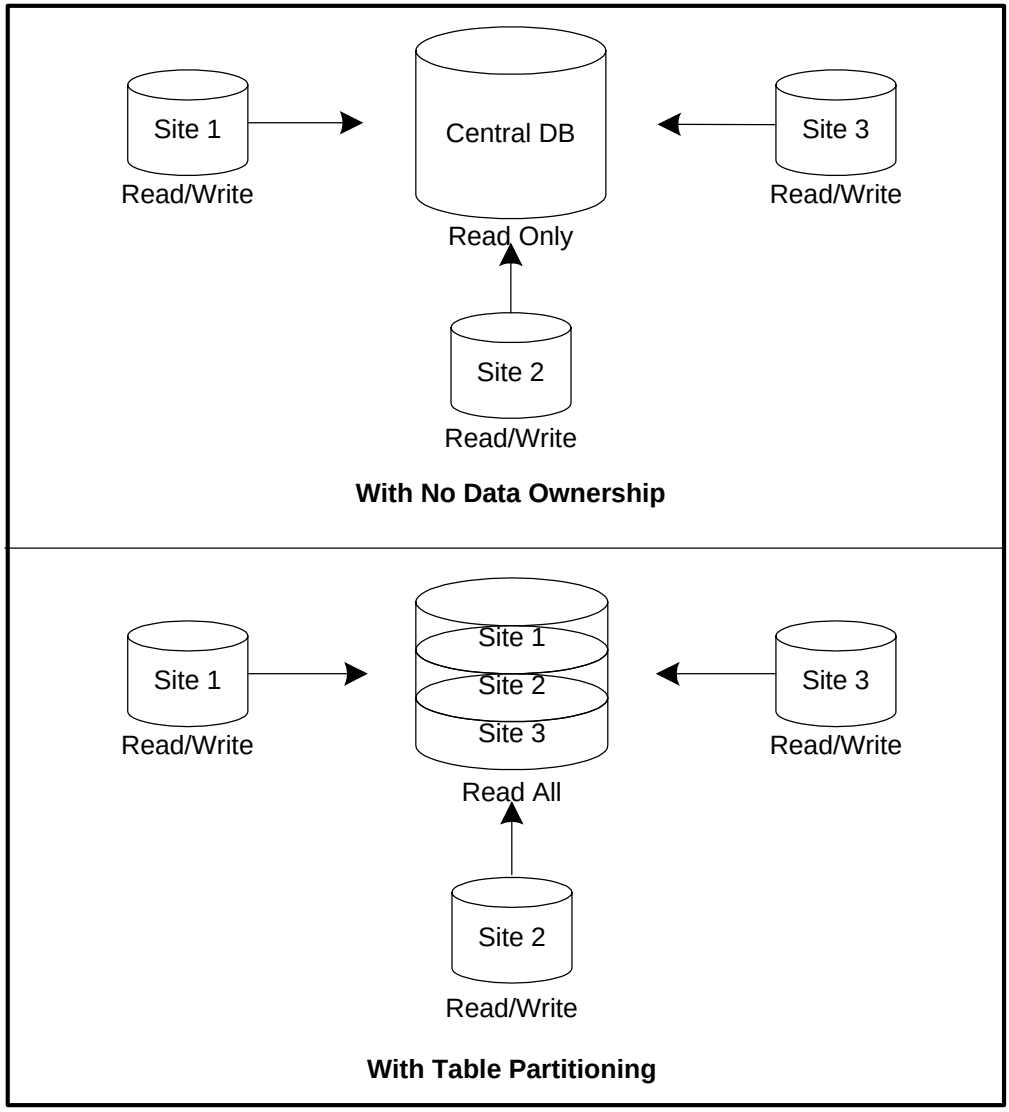


Figure 15–2: Data Consolidation Models

Peer-to-peer Model

In a *peer-to-peer* model (or “update anywhere”) any user at any site can update data. This is the most flexible replication model. However, in a peer-to-peer scheme, data collision is a side effect that must be addressed. Data collision must be resolved based on business requirements. An approach to data collision resolution is discussed later in this chapter.

Figure 15–3 illustrates the peer-to-peer model.

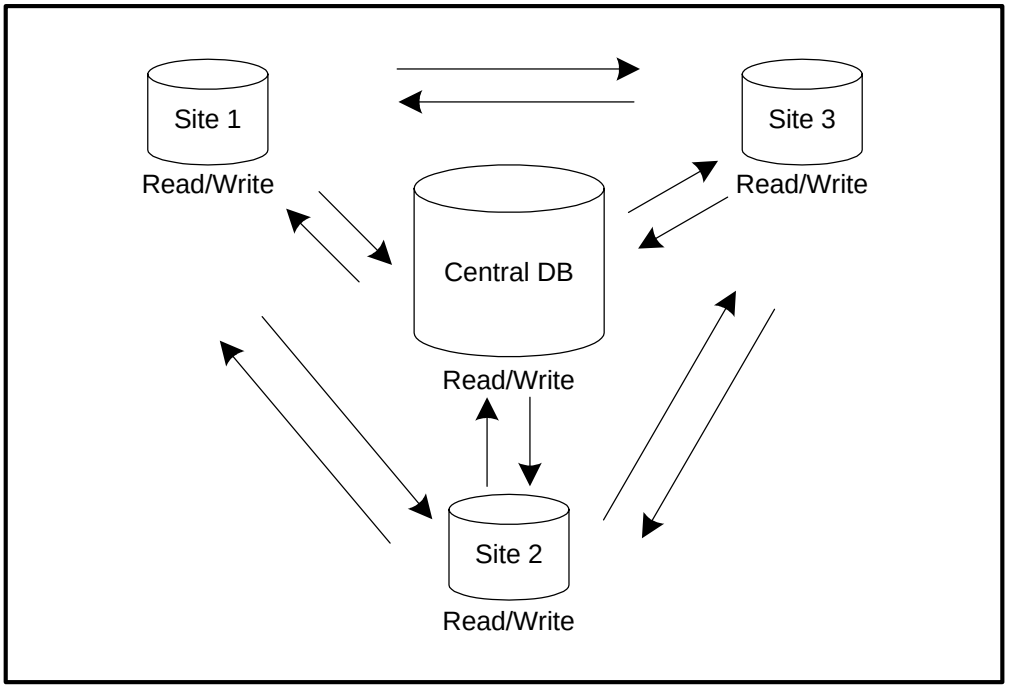


Figure 15–3: Peer-to-peer Model

15.3 Implementing Log-based Site Replication

With log-based site replication, a central database is replicated in its entirety to one or more secondary sites. Log-based site replication allows the following:

- The creation of “hot” standby sites in case the primary site fails
- Flexible snapshot capability to control the timeliness of replication
- A transparent method of maintaining a remote backup of a primary database

15.3.1 Log-based Replication Procedure

Site replication with Progress databases is implemented through the use of after-imaging and database backup procedures. For complete information about these procedures, see Chapter 7, “Backing Up a Database” and Chapter 12, “Using Two-phase Commit.”

Follow these steps to implement log-based site replication:

1 ♦ Add after-imaging extents and then enable after-imaging in the primary database. For information about after-imaging extents and enabling after imaging, see Chapter 11, “After-Imaging.”

2 ♦ Use the PROSTRCT utility with the LIST option to create a structure description file containing the central database’s data structure. For information about the structure description file and the PROSTRCT utility, see Chapter 9, “Maintaining Database Structure.”

3 ♦ With the structure description file produced from the central database, use PROSTRCT with the CREATE option to create an additional database on the remote system.

4 ♦ Perform a backup of the primary database to initialize the secondary database. This step creates a basis for subsequent roll-forward operations. For information about performing a backup, see Chapter 7, “Backing Up a Database.”

5 ♦ Restore the backup copy of the primary database to the secondary database.

6 ♦ Use the RFUTIL command with the option EXTENT FULL to monitor the after-image extents. This will automatically determine which image extent is ready for replication (or transfer) to the secondary site. You can transfer the after-image extent file to the secondary site using an OS command to remote copy .

For more information about RFUTIL, see Chapter 19, “Database Administration Utilities.”

7 ♦ Once the after-image extent has been transferred to the secondary site, use RFUTIL with the EMPTY option to mark the extent “empty” and ready for use on the primary database.

8 ♦ Implement a process to monitor and transfer full after-image extents (AI extents). You can copy AI extents to an AI log, then transfer the log contents to the secondary site on a continuous basis.

9 ♦ If it becomes necessary to shift operations to the secondary site, transfer the last “full” and “busy” after-image extents and roll-forward to completion. Start up of the secondary site database causes the database to undergo crash recovery, resulting in the shift to the secondary site.

For more information about performing roll-forward recovery, see Chapter 8, “Recovering a Database.”

Figure 15–4 shows after-image extent files replicated from the primary to a secondary database.

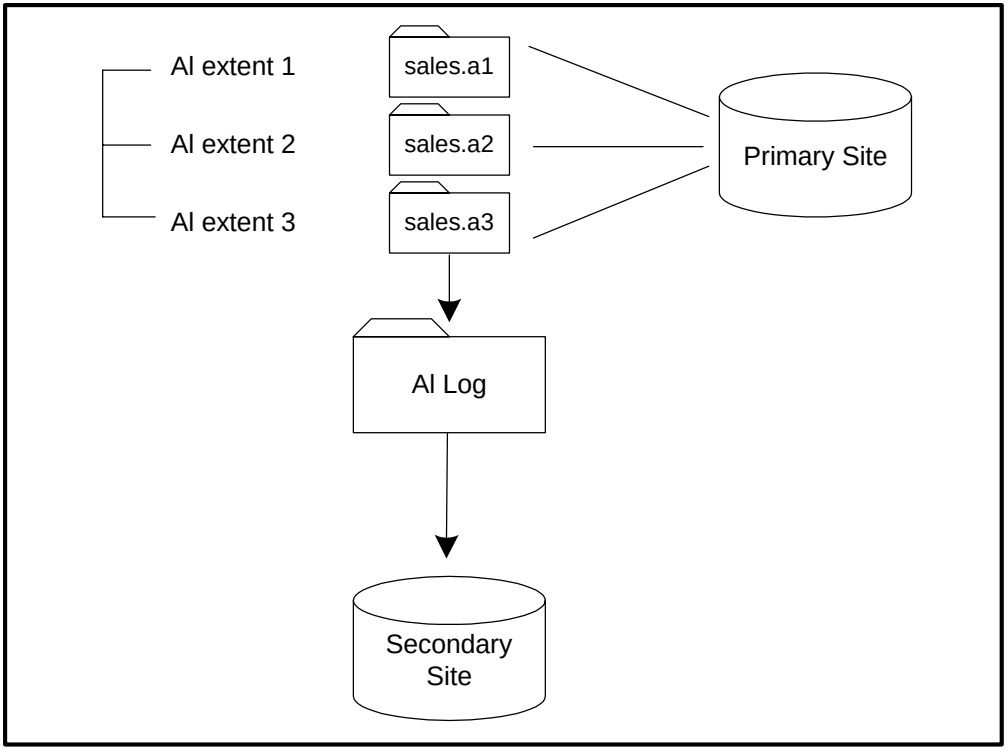


Figure 15–4: Example Of Site Replication

16

Using the Event Log

The Progress Version 9 database engine logs significant database events such as startup parameter settings, startup, shutdown, and system error messages, and application-related events. This chapter details the messages written to the event log.

Specifically, this chapter contains the following sections:

- Progress Version 9 Event Log File
- Managing the Event Log File Size
- Event Logging On Windows

16.1 Progress Version 9 Event Log File

The Progress event log is a text file that contains a history of significant database events, such as Progress startup parameter settings and startup, shutdown, and system error messages. This file has a .lg extension. Entries in the event log can help you trace events preceding database crashes.

16.2 Managing the Event Log File Size

The event log (LG) file expands as you use the database. If it becomes too large, you can reduce its size by removing old log entries. To remove log entries from an LG file, use the Progress Log Maintenance (PROLOG) utility or a text editor. Do not remove entries from the log file while the database is in use.

Enter the following command to remove entries from an event log file:

```
prolog database-name
```

The PROLOG utility removes all but the most recent entries from the log file. For more details, see the description of the PROLOG utility in Chapter 19, "Database Administration Utilities."

16.3 Event Logging On Windows

In addition to the Progress event log, the Progress Server writes events to the Windows Event Log. The *Event Log* is the object that enables Windows users to view the status of application, security, and system processes, and to view their associated events. The Progress database is an application process that writes events to the Application Event Log. You use the Event Viewer to see the Event Log's contents. You can customize the Event Viewer so that it displays only the event types that you want to view. You access the Event Viewer through the Administrative Tools program group. For a list of Windows operating systems currently supported by Progress, see the *Progress Installation and Configuration Guide Version 9 for Windows*.

The components that enable the Progress service to log messages to the Application event log database are described in Table 16–1.

Table 16–1: The Progress Event Logging Components

This component . . .	performs this function . . .
The Event Viewer	Enables users to view the Event Log.
The Event Log	Records event information.
PROMSGS.DLL	Contains Progress database messages for the event log.
CATEGORY.DLL	Contains the 14 categories into which Progress database messages might fall.
PROMSGS file	Contains the full set of the Progress database messages including translated versions. The PROMSGS file is installed to the directory with the Progress executable.

16.3.1 Managing Progress Events On Windows

You can define the level of event logging that you want your Progress application to support by using the Event Level Environment Variable (EVTLEVEL), or the Event Level startup parameter (`-evtlevel`). Use the Progress Control (PROCONTROL) utility to supply the Event Level as a startup parameter or environment variable. See Chapter 19, "Database Administration Utilities," for information about the PROCONTROL utility.

Table 16–2 lists valid event level values.

Table 16–2: Event Level Values

Value	Description
None	No Progress events are written to the event log.
Brief	Progress messages defined as Error and Warning messages are written to the event log.
Normal	Progress messages defined as Error and Warning messages are written to the event log. In addition, any Progress message that is normally written to the log file (.lg) is also written to the Event Log. This is the default.
Full	Every message generated by Progress is written to the Event Log. Any Progress messages generated using the Message Statement are also written to the log file.

16.3.2 Understanding the Event Log Components

The components of the Event Log are standards defined by Windows. Figure 16–1 illustrates the Event Log components when shown through the Event Viewer.

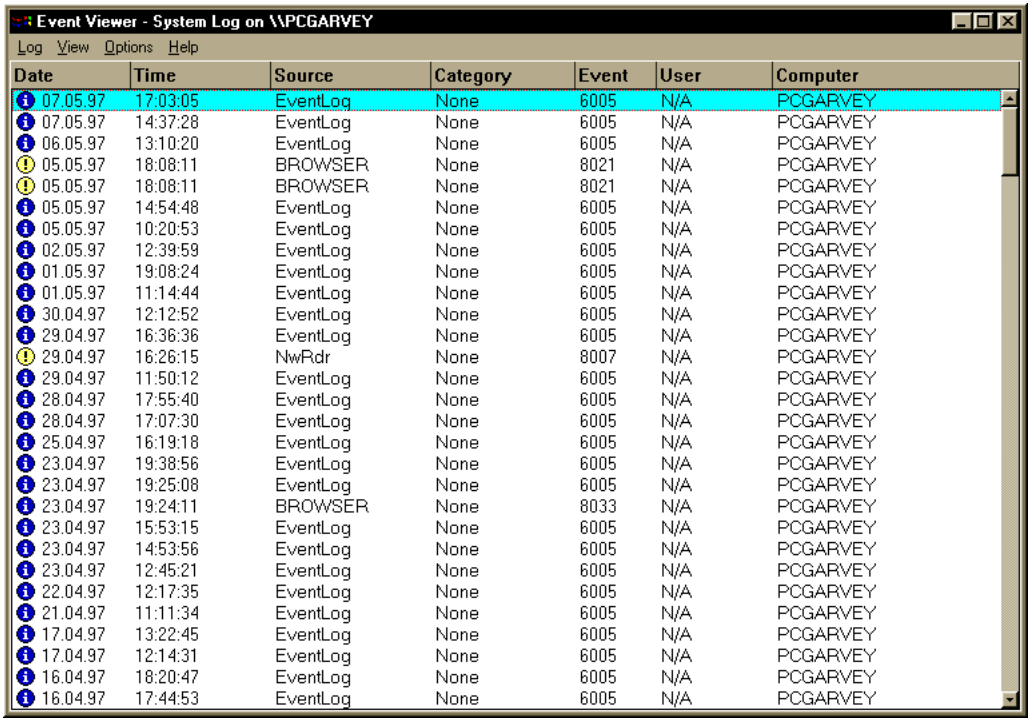


Figure 16–1: NT Event Log Components

Table 16–3 describes how Progress uses the Event Log columns.

Table 16–3: How Progress Uses Event Log Components *(1 of 2)*

This column . . .	includes this information . . .
Date	Date the event occurred.
Time	Time the event occurred.

Table 16–3: How Progress Uses Event Log Components *(2 of 2)*

This column . . .	includes this information . . .
Source	Source of the event. This is the name of the connected Progress database, if a database is connected. If no database is connected, then "Progress" is listed. If you are using the Progress AppServer, "Progress" is also the default source for Progress AppServer messages; however, you can override the default source name by specifying the `-logname` application broker startup parameter.
Category	To help you isolate the cause of the message displayed in the Event Log, Progress supports 16 event categories. The event categories are: AIW, APPBROKER, APPSERVER, APW, BACKUP, BIW, DATASERVER, MON, OIBRKR, OIDRVR, Progress, RFUTIL, SERVER, SHUT, USER, and WDOG. When no database is connected, Progress is specified as the category. The APPBROKER and APPSERVER categories appear in the Event Log only when messages are logged by the Progress AppServer application broker and application server, respectively. All categories reside in a file called `category.dll`. These categories correspond to the existing categories of events that are displayed in the `progress.lg` file (application broker and application server events are displayed in the AppServer log file, `proapsv.lg`). (Note that DATASERVER is not included as a category in the standard `progress.lg` file.)
Event	Associates to the Progress message that was generated. These are the same message numbers that are displayed in the standard database .lg file.
User	Identifies the user logged in to the Windows workstation where the event occurred.
Computer	Identifies the name of the Windows workstation where the event occurred. The Event Viewer enables you to get more information about events by double-clicking on any event.

You can view additional information about an event by double-clicking on it. Windows displays the Event Detail screen, as shown in Figure 16–2.

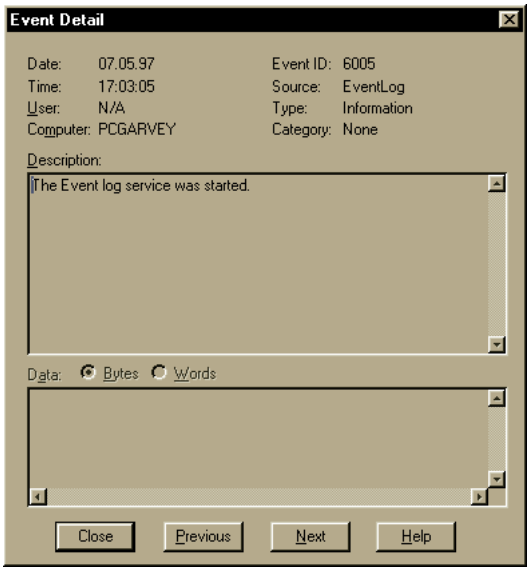


Figure 16–2: NT Event Detail Dialog Box

16.3.3 The Event Log and the Registry

Windows requires that applications using the Event Log be bound to all of the necessary components. This means that the PROMSGS.DLL and the CATEGORY.DLL must be bound to any Progress database. The database engine stores this information in the registry. The engine makes the registry entries and performs any binding operations that are necessary when you initially access a database. When the engine binds the .DLL files to the database, it writes the fully qualified pathname to the registry. If you delete the database, you must manually remove the associated data from the registry. Or, if you move the location of the .DLLs after you access the database, you must manually edit the registry data. The Progress components can be found in the following location in the registry:

```
HKEY_LOCAL_MACHINE
SYSTEM
 CurrentControlSet
  Services
   EventLog
    Security
    System
    Application
     PROGRESS
     <Database Name>
```

See the Microsoft Windows documentation for more information about editing registry files.

When the database engine tries to find the .DLLs before this information is included in the registry, it performs the search according to these rules:

1. Search the current directory.

2. If the .DLL is not in the current directory, the engine searches the directory where the Progress executable is located.

3. If the .DLL is not in the same directory as the Progress executable, the engine searches the user's path.

If the .DLL is not in the user's path, the engine generates a message stating that the .DLL cannot be found, and it writes a message to the Progress event log file.

PART III

Reference

Startup and Shutdown Commands

Database Startup Parameters

Database Administration Utilities

Virtual System Tables

17

Startup and Shutdown Commands

This chapter describes the Progress database startup and shutdown commands in alphabetical order. It describes the purpose, syntax, and primary parameters for each command. For a complete list and description of all parameters you can specify with these commands, see Chapter 18, “Database Startup Parameters.”

This chapter contains the following sections:

- Startup Command Syntax
- Database Startup and Shutdown Commands

17.1 Startup Command Syntax

Figure 17–1 shows the conventions used in command syntax descriptions.

```
command  [ db-name ] [ parameter ] [ value ] . . .
```

Figure 17–1: Syntax Conventions

For example, the following command allows 100 users to access the sports database and then set values for the database connection, performance, and network parameters.

```
proserve sports -n 100 -B 30000 -L 1000 -S sprtsv -H sys27
```

Table 17–1 describes each of the command components.

Table 17–1: Progress Command Components

Component	Description
command	On UNIX, the command runs a script that executes a Progress executable with appropriate parameters. On Windows, some commands run a batch file that executes a Progress executable with appropriate parameters. Other commands run a Progress executable directory.
db-name	Name of the database you want to connect to.
parameter, qualifier	Operating criteria for the command.
value	Numeric value or file specification for the parameter.

NOTE: On both UNIX and Windows, enter parameters exactly as shown in the syntax descriptions.

17.2 Database Startup and Shutdown Commands

Database startup commands start Progress database processes. Table 17–2 summarizes the tasks performed by each command.

Table 17–2: Database Startup and Shutdown Commands

Task	Command
Start a Progress server-group.	`proserve` `-servergroup` *servergroup-name*
Start a server or broker for a multi-user Progress database.	`proserve` *db-name* `-S` *service-name* `-H` *host-name* `-N` *network-type*
Shut down a multi-user server or broker for a Progress database.	`proshut` *db-name*
Start an asynchronous page writer (APW) for a database.[1]	`proapw` *db-name*
Start a before-image writer (BIW)[1].	`probiw` *db-name*
Start an after-image writer (AIW)[1].	`proaiw` *db-name*
Stop all writes to database files by enabling a "quiet" processing point.	`proquiet` *db-name parameter*
Start the Progress Watchdog utility[1].	`prowdog` *db-name*
Shut down a remote Progress DataServer.	`proshut` *db-name* `-S` *service-name* `-H` *host-name* `-N` *network-type*
Shut down an APW, AIW, BIW, or Watchdog process[1].	`proshut` *db-name* Choose option 1 (Disconnect a User) to disconnect the process.

[1] Option available only on Enterprise Database product.

PROAIW Command

Starts the after-image writer (AIW) process. The AIW improves performance by writing notes to the after-imaging file. For more information on the AIW, see Chapter 14, "Managing Performance."

SYNTAX

Operating System	Syntax
UNIX Windows	`proaiw` *db-name*

db-name

Specifies the database where you want to start the AIW process.

NOTES

- To stop the AIW, disconnect it by using the PROSHUT command. You can start and stop the AIW at any time without shutting down the database.
- The AIW counts as one user. You might have to increment the value of the Number of Users (`-n`) parameter to allow for the AIW. However, the AIW does not count as a licensed user.
- You can increase the number of buffers in the after-image buffer pool by using the After-image Buffers (`-aibufs`) parameter. Increasing the number of buffers when running an AIW increases the availability of empty buffers to client and server processes. Increasing the After-image Buffers parameter has no effect if the AIW is not running.
- After-image writers are only available on Enterprise systems.

PROAPW Command

Starts an asynchronous page writer (APW). APWs improve database performance by performing overhead operations in the background. For more information on APWs, see Chapter 14, "Managing Performance."

SYNTAX

Operating System	Syntax
UNIX Windows	`proapw` *db-name*

db-name

Specifies the database where you want to start an APW.

NOTES

- To stop an APW, disconnect it by using the PROSHUT command. You can start and stop APWs at any time without shutting down the database.
- Each APW counts as a user. You might have to increase the value of the Number of Users (`-n`) parameter to allow for APWs. However, APWs do not count as licensed users.
- The optimal number depends on your application and environment. To start, use one page writer for each disk where the database resides. If data gathered from PROMON indicates that this is insufficient, add more. For more information on PROMON see the "PROMON Utility" section in Chapter 19, "Database Administration Utilities."
- For an application that performs many updates, start one APW for each disk containing your database, plus one additional APW. Applications that perform fewer changes to a database require fewer APWs.
- Asynchronous page writers are only available on Enterprise systems.

PROBIW Command

Starts a before-image writer (BIW) process. The BIW improves database performance by performing before-image overhead operations in the background. For more information on the BIW, see Chapter 14, “Managing Performance.”

SYNTAX

Operating System	Syntax
UNIX Windows	`probiw` *db-name*

db-name

Specifies the database where you want to start a BIW.

NOTES

- To stop the BIW process, disconnect it by using the PROSHUT command. You can start and stop the BIW at any time without shutting down the database.
- The BIW process counts as one user. You might have to increment the value of the Number of Users (`-n`) parameter to allow for the BIW. However, the BIW does not count as a licensed user.
- You can increase the number of before-image buffers with the Before-image Buffers (`-bibufs`) parameter. Increasing the number of buffers increases the availability of empty buffers to client and server processes.

PROQUIET Command

Stops all writes to database files by enabling a "quiet" processing point (useful for advanced backup strategies). You can also use the PROQUIET command with the bithreshold parameter to adjust the size of the recovery log threshold online. Use the PROSERVE command with the `-bithold` startup parameter to set the size of the primary recovery log threshold on startup.

For more information on using database quiet points, see Chapter 7, "Backing Up a Database," and Chapter 14, "Managing Performance."

SYNTAX

Operating System	Syntax
UNIX Windows	`proquiet` *dbname* `-C { { enable | disable } | bithreshold` *n* `}`

dbname

Specifies the name of the database where you are enabling or disabling a quiet processing point.

`enable | disable`

Enables or disables a quiet processing point. Any processes that attempt transactions while a quiet point is enabled must wait until the quiet point is disabled.

`bithreshold` *n*

Specifies the maximum size to which BI recovery files can grow, where *n* is an integer specifying the size of the threshold in MB. You can increase the size of the threshold above the current value or reduce the size to one cluster larger than the size of the recovery log file at the time the PROQUIET command is issued.

NOTE: Though the above table lists the `-C` parameter to show the complete syntax, you do not need to use the `-C` parameter in the PROQUIET syntax as PROQUIET calls `-C` for you.

EXAMPLES

- Use the PROQUIET command to manage the primary recovery area (BI) threshold before your database stalls.

  For example, to start a server and set a 500MB BI threshold, allowing the system to stall if that threshold is reached, use the PROSERVE command as follows:

```
proserve mydemo -bithold 500 -bistall
```

  Assume a long running transaction causes the expansion of the BI, but before the threshold is reached you receive the following message:

```
BI file size has grown to within 90% of the threshold value 523763712.
(6559)
```

  After receiving message 6559, you decide to increase the BI threshold to 1GB while the database remains online and investigate the cause of the unexpected BI growth before the system stalls. Use the PROQUIET command, as follows:

```
proquiet mydemo bithreshold 1000
```

  The above command establishes a quiet point and increase the threshold. The database does not stall.

  NOTE: In practice, invoke PROQUIET commands by using a script so that they occur as quickly as possible, and with the least amount of impact to the online system.

- When a database stalls because the BI threshold is reached, the stall causes an implicit quiet point and the database engine writes a message to the log file. To expand the BI threshold and continue forward processing, use the PROQUIET command with the `bithreshold` parameter only:

```
proquiet mydemo bithreshold 1000
```

PROSERVE Command

Starts the broker, which in turn spawns the server. The server process coordinates all access to the specified Progress database.

SYNTAX

Operating System	Syntax
UNIX Windows	`proserve { db-name | -servergroup [ servergroup-name ]} [ parameters ]`

`-servergroup` *servergroup-name*

Specifies the logical collection of server processes to start. The *servergroup-name* you specify must match the name of a servergroup in the `conmgr.properties` file. You create servergroups using the Progress Explorer Database Configuration Tools, which saves them in the `conmgr.properties` file.

db-name

Specifies the specific database you want to start.

parameters

Specifies the startup parameters for the broker/server. See Chapter 19, "Database Administration Utilities," for a list of broker/server startup parameters.

NOTES

- You can specify only one database name when using PROSERVE to start a broker or servergroup.
- Servergroups manage network connections four separate ways:
  - Accept no network connections
  - Accept SQL-92 and 4GL network connections
  - Accept only SQL-92 network connections
  - Accept only 4GL network connections

- Typically, servergroups share common attributes such as connection port, number of servers, and how connected clients are distributed among the severs.
- You create servergroups using the Progress Explorer Database Configuration Tool, which saves them in the `conmgr.properties` file. The *`servergroup-name`* you specify with the PROSERVE `-servergroup` parameter must match the name of a servergroup in the `conmgr.properties` file. Do not edit the `conmgr.properties` file directly. Instead, use Progress Explorer. For more information on the Progress Explorer, click the help icon within the Progress Explorer application.
- The behavior of the `-servergroup` parameter is similar to the behavior of the `-pf` (parameter file) parameter. In effect, `-servergroup` causes a server to load the parameters associated with the servergroup, including the database name.

  It is possible to override the parameter values associated with a servergroup by adding additional parameters to the PROSERVE command. For example, if the database buffer pool is set to 10,000 within the configuration associated with a servergroup, you can specify a larger value by adding an additional parameter:

```
proserve -servergroup sports2000.myconfig.4GLdefault -B 20000
```

  Conversely, if you specify a startup parameter before the `-servergroup` parameter, the startup parameter can be overridden when the same parameter is set in the servergroup configuration file. For example, if you place the additional parameter before the `-servergroup` parameter, the database buffer pool remains 10,000:

```
proserve -B 20000 -servergroup sports2000.myconfig.4GLdefault
```

PROSHUT Command

Shuts down the Progress database server and individual processes. Before you shut down the broker, have all application users quit their Progress sessions. If necessary, you can disconnect users by using the PROSHUT command's Disconnect a User or Unconditional Shutdown parameters.

SYNTAX

Operating System	Syntax
UNIX Windows	proshut *db-name* [-b | -by | -bn | -C list | -C disconnect *usernum* | -F | -Gw | -H *host-name* | -S *service-name* | -cpinternal *codepage* | -cpstream *codepage*] ...

db-name

Specifies the database the server is running against.

-b

Directs PROSHUT to perform a batch shutdown. When no client is connected, the database automatically shuts down. When one or more clients are connected, PROSHUT prompts the user to enter "yes" to perform an unconditional batch shutdown and to disconnect all active users; or "no" to perform a batch shutdown only if there are no active users. The -b parameter combines the functionality of the -by or -bn parameters.

`-by`

Directs PROSHUT to perform an unconditional batch shutdown and to disconnect all active users. Note that PROSHUT does not give users any notice before disconnecting them.

`-bn`

Directs PROSHUT to perform a batch shutdown only if there are no active users.

`-C list`

Lists all of the users connected to the database. The list is printed out to the screen without any page breaks.

`-C disconnect` *`usernum`*

Allows you to initiate a disconnect for the specified user. This is similar to option 1 of the PROSHUT menu.

`-F`

Starts an emergency shutdown. To use this parameter, you must run PROSHUT on the machine where the server resides. This parameter is not applicable for remote shutdowns or DataServer shutdowns.

`-Gw`

For DataServers, specifies the DataServer broker to shut down.

`-H` *`host-name`*

Specifies the machine where the database server runs. If issuing the shutdown command from a remote machine, specify the host name.

`-S` *`service-name`*

Specifies the database server or broker process. If issuing the shutdown command from a remote machine, specify the service name.

`-cpinternal` *`codepage`*

An internationalization startup parameter that identifies the code page used in memory.

`-cpstream` *`codepage`*

An internationalization startup parameter that identifies the code page used for stream I/O.

When you enter the PROSHUT command without the -by, -bn, or -F parameters, the following menu appears:

```
1 Disconnect a User
2 Unconditional Shutdown
3 Emergency Shutdown (Kill All)
x Exit
```

The following table lists the menu options and their actions:

Option	Action
1	Prompts you for the number of the user you want to disconnect.
2	Disconnects all users and shuts down the database.
3	Prompts you to confirm your choice. If you cancel the choice, you cancel the shutdown. If you confirm the choice, PROSHUT displays the following message: Emergency shutdown initiated . . . PROSHUT marks the database for abnormal shutdown, kills all remaining processes connected to the database, and deletes shared-memory segments and semaphores. The database is in a crashed state. PROSHUT performs normal crash recovery when you restart the database and backs out any active transactions.
x	Cancels the shutdown without taking any action.

EXAMPLES

- You can shut down using PROMON's "Shut Down Database" menu option.
- The user who shuts down the server must have started it, or be root (on UNIX).
- When you initiate PROSHUT over a network, the amount of time that it takes to actually shut down all of the Progress processes and to free any ports varies depending on the number of clients, brokers, and servers that must be shut down. The PROSHUT command might return control to the terminal before all of the processes are stopped and resources are freed.
- If you specified a unique value for `-cpinternal` or `-cpstream` when you opened the database, you must specify that same value for `-cpinternal` or `-cpstream` when you close the database with the PROSHUT command. If you do not, PROSHUT uses the values for `-cpinternal` and `-cpstream` found in the main startup parameter file created during installation (such as `DLC/startup.pf`). If the values of `-cpinternal` or `-cpstream` specified for your database do not match the values specified in the main startup parameter file, you receive the following error message:

```
Code page conversion table for table-name to table-name was not found.
(6063)
```

PROWDOG Command

Starts the Progress Watchdog process. The Watchdog cleans up after improperly terminated processes. For example, it releases any locks or shared-memory structures that those processes might hold.

SYNTAX

Operating System	Syntax
UNIX Windows	`prowdog` *db-name*

db-name

Specifies the database to clean up after.

NOTES

- If the Watchdog finds a process that is no longer active, it releases all the appropriate record locks, backs out any live transactions, releases any shared-memory locks, and closes the connection. If the lost process is a server, it disconnects and cleans up all appropriate remote clients.
- If the process was changing shared memory when it terminated, shared memory is in an inconsistent state; the Watchdog forces a shutdown to protect the database.
- The Watchdog cannot detect lost remote clients because remote clients are not associated with a process. Instead, a network protocol timeout mechanism notifies the server that the network connection was lost.

18

Database Startup Parameters

This chapter describes Progress database server startup parameters. They are presented in quick reference tables in the beginning of this chapter. Then, each startup parameter is described in detail and listed alphabetically by syntax. The syntax of the parameters is the same for UNIX and Windows unless otherwise noted.

Specifically, this chapter contains the following sections:

- Issuing Startup Parameters
- Database Server Performance Parameters
- Database Server-type Parameters
- Database Server Internationalization Parameters
- Database Server Statistics Parameters
- Database Server Network Parameters
- Alphabetical Listing Of Database Startup Parameters

For a description of all the Progress startup parameters, including those not database related, see the *Progress Startup Command and Parameter Reference*.

18.1 Issuing Startup Parameters

You can change the default startup parameters by using startup parameters on a command line, or by incorporating them into a script. You can also use a Progress *parameter file*.

A parameter file can include any number of startup parameters. This is especially useful if you regularly use the same parameters, or if more parameters are required than can easily fit on the command line. To identify a parameter file, you use the Parameter File (-pf) parameter, which has the following syntax:

SYNTAX

```
-pf filename
```

You can also include the -pf parameter in the parameter file to reference another parameter file.

NOTE: If duplicate startup parameters are read from the startup line or .pf file, the last duplicate parameter read takes precedence.

18.2 Database Server Performance Parameters

Use the parameters listed in Table 18–1 to optimize server performance.

Table 18–1: Server Performance Parameters *(1 of 3)*

Parameter	Syntax	Purpose
After-image Buffers	-aibufs *n*	Specifies the number of after-image buffers when running AIW.
After-image Stall	-aistall	Suspends database activity when an empty after-image (AI) file is unavailable.
Blocks in Database Buffers	-B *n*	Specifies the number of blocks in the database buffers.
Before-image Buffers	-bibufs *n*	Specifies the number of before-image buffers when running BIW.
Threshold Stall	-bistall	Quiets a database and sends a message to the log file when the recovery log threshold is reached. Use with -bithold.
Recovery Log Threshold	-bithold *n*	Specifies the maximum size of the recovery log files in MB.

Table 18–1: Server Performance Parameters *(2 of 3)*

Parameter	Syntax	Purpose
Direct I/O	`-directio`	Opens all files in unbuffered mode.
Event Level	-evtlevel	Specifies the level of information written to the NT Application Event Log.
Before-image Cluster Age	`-G` *n*	Specifies the number of seconds before Progress reuses a before-image cluster.
Group Delay	`-groupdelay` *n*	Specifies the number of milliseconds a transaction waits before committing.
Hash Table Entries	`-hash`	Specifies the number of hash table entries for the buffer pool.
No Crash Protection	`-i`	Runs Progress without using database integrity or recovery.
Lock Table Entries	`-L` *n*	Specifies the number of entries in the record locking table.
Delayed BI File Write	`-Mf` *n*	Delays writing the last before-image (BI) file records.
VLM Page Table Entry Optimization[1]	`-Mpte`	Allocates shared memory in multiples of 8 MB for VLM64 support.
Shared-memory Overflow Size	`-Mxs` *n*	Replaces the default value of the shared-memory overflow area.
Number of Users	`-n` *n*	Specifies the maximum number of users connected to the database.
Pin Shared Memory	`-pinshm`	Prevents the database engine from swapping shared memory contents to disk.
Semaphore Sets[2]	`-semsets` *n*	Changes the number of semaphore sets available to a broker.

Table 18–1: Server Performance Parameters *(3 of 3)*

Parameter	Syntax	Purpose
Spin Lock Retries	`-spin` *n*	Specifies the number of times a process tries to acquire a latch before pausing.

[1] Compaq Tru64 only.

[2] UNIX only.

18.3 Database Server-type Parameters

Use the parameters listed in Table 18–2 to start a particular type of server.

Table 18–2: Server-type Parameters

Parameter	Syntax	Purpose
Auto Server	`-m1`	Starts an auto server. Used internally by the Database broker.
Manual Server	`-m2`	Manually starts a server after you start a broker.
Secondary Login Broker	`-m3`	Starts a secondary broker.

18.4 Database Server Internationalization Parameters

Use the parameters listed in Table 18–3 to control the format in which data appears.

Table 18–3: Server Internationalization Parameters *(1 of 2)*

Parameter	Syntax	Purpose
Conversion Map	`-convmap` *filename*	Identifies the conversion map file.
Case Table	`-cpcase` *tablename*	Identifies the case table that establishes case rules for the code page.

Table 18–3: Server Internationalization Parameters *(2 of 2)*

Parameter	Syntax	Purpose
Collation Table	`-cpcoll` *`tablename`*	Identifies a collation table to use with the code page.
Internal Code Page	`-cpinternal` *`codepage`*	Identifies the code page that Progress uses in memory.
Log File Code Page	`-cplog` *`codepage`*	Identifies the code page used for writing messages to the log file.
Print Code Page	`-cpprint` *`codepage`*	Identifies the code page used for printer output.
R-code in Code Page	`-cprcodein` *`codepage`*	Identifies the code page used for reading r-code text segments.
Stream Code Page	`-cpstream` *`codepage`*	Identifies the code page used for stream I/O.
Terminal Code Page	`-cpterm` *`codepage`*	Identifies the code page for character terminals.

18.5 Database Server Statistics Parameters

Use the parameters listed in Table 18–4 to collect statistics for table and index access.

Table 18–4: Server Statistics Collection Parameters *(1 of 2)*

Parameter	Syntax	Purpose
Base Index	`-baseindex` *`n`*	Specifies a range of indexes for which you want to collect statistics. Use with `-indexrangesize`.
Base Table	`-basetable` *`n`*	Specifies a starting table number in a range of tables for which you want to track access statistics. Use with `-tablerangesize`.

Table 18–4: Server Statistics Collection Parameters *(2 of 2)*

Parameter	Syntax	Purpose
Index Range Size	`-indexrangesize` *n*	Specifies the number of indexes to track for access statistics.
Table Range Size	`-tablerangesize` *n*	Specifies the number of tables for which you want to collect statistics.

18.6 Database Server Network Parameters

Use the parameters listed in Table 18–5 to supply the broker with necessary network information.

Table 18–5: Server Network Parameters *(1 of 2)*

Parameter	Syntax	Purpose
AdminServer Port	`-adminport { service-name | port }`	Connects a servergroup and an AdminServer.
SQL-92 Server Java Classpath	`-classpath` *pathname*	Identifies the Java classpath to use when starting an SQL server.
Host Name	`-H` *host-name*	Specifies a remote host.
Maximum Clients Per Server	`-Ma` *n*	Specifies the maximum number of remote users per database server.
Maximum Dynamic Server	`-maxport` *n*	Specifies the highest accessible port number in a specified range.
Minimum Clients Per Server	`-Mi` *n*	Specifies the number of remote users on a server before a broker starts another server.
Minimum Dynamic Server	`-minport` *n*	Specifies the lowest accessible port number in a specified range.

Table 18–5: Server Network Parameters *(2 of 2)*

Parameter	Syntax	Purpose
Maximum Servers	-Mn *n*	Specifies the maximum number of remote client servers that a broker can start.
Servers Per Protocol	-Mp *n*	Specifies the maximum number of servers to serve remote users for a protocol.
Maximum Servers Per Broker	-Mpb *n*	Specifies the maximum number of servers that multiple brokers can start to serve remote users for a protocol.
Network Type	-N *network-type*	Identifies the network communications protocol.
Configuration Properties File	-properties *filename*	Identifies the properties file an AdminServer uses when starting a database server or servergroup.
Service Name	-S { *service-name* | *port-number* }	Specifies the service or port number to be used by a broker process.
Server Group	-servergroup *name*	Identifies a logical collection of server processes to start.

18.7 Startup Parameter Usage Categories

Startup parameters are organized into usage categories. Table 18–6 describes each usage category.

Table 18–6: Startup Parameter Categories

Usage Type	Used to. . .
Client Session (CS)	Start a client session.
Client Connections (CC)	Connect to a specific database.
Progress Database Server (DBS)	Start a server or broker.
DataServer (DS)	Start non-Progress DataServers.
Open Interface Driver (OID)	Start the Open Interface Driver.

NOTE: Some parameters fall into more than one category.

The following sections focus on Client Connection (CC) and Database Server (DBS) parameters. For a description of all CS, CC, DBS, DS, and OID parameters, see the *Progress Startup Command and Parameter Reference*.

18.8 Alphabetical Listing Of Database Startup Parameters

This section describes the Progress database server startup parameters in detail, in alphabetical order by syntax. Each description begins with a table that shows the syntax of the parameter, then provides other information about it.

18.8.1 AdminServer Port (-adminport)

Operating System and Syntax	UNIX Windows	-adminport *service-name port*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	–	–	–	–

service-name

The name of the service to be used by the AdminServer.

port

The port number the AdminServer uses to communicate with servergroups. The default port is 7832.

Use AdminServer Port (`-adminport`) to establish communication between a servergroup and an AdminServer. The AdminServer uses this parameter internally. The `-adminport` setting must match the `-admin` setting specified when the AdminServer was started.

18.8.2 After-image Buffers (-aibufs)

Operating System and Syntax	**UNIX Windows**	`-aibufs` *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	–	1	–	5

n

The number of after-image buffers.

Use After-image buffer (`-aibufs`) to specify the number of after-image buffers. This parameter is useful only when running the after-image writer (AIW) because the AIW writes the filled after-image buffers to disk, making the buffers available to other client and server processes. Progress Software recommends setting `-aibufs` to a value of 1.5 times the value of the Before-image Buffers (`-bibufs`) parameter, or a minimum of 5.

Without the AIW writing the buffers, any gain from increasing the number of buffers is negligible.

18.8.3 After-image Stall (-aistall)

Operating System and Syntax	UNIX Windows	-aistall		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	–	–	–	–

Use After-image Stall (-aistall) to suspend database activity if all AI files are filled. -aistall ceases all database activity and continues to send the following message to the log file until the AI extent is emptied:

```
Can't switch to after-image extent filename it is full. (3775)
Backup ai extent and mark it as empty (3776)
```

When using after-image (AI) files, monitor the status of the files to ensure that the AI extents do not run out of space and cause the database to hang. Without the use of -aistall, the database shuts down when the AI files are filled.

18.8.4 Blocks In Database Buffers (-B)

Operating System and Syntax	UNIX Windows	-B *n*		
Use With	**Maximum Value**[1]	**Minimum Value**	**Single-user Default**	**Multi-user Default**
CC, DBS	System dependent[1]	10	20	(8 * *users*) [2]

[1] Limited by available memory.

[2] The *users* value is specified by the Number of Users (-n) parameter.

n

The number of blocks in the database buffers.

Use Database Buffers (-B) to specify the number of blocks in the database buffers. The optimum value depends on your application.

18.8.5 Base Index (-baseindex)

Operating System and Syntax	UNIX Windows	-baseindex *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	–	–	–	–

n

The starting index number in the range of indexes for which you want to track access statistics.

Use Base Index (-baseindex) with Index Range Size (-indexrangesize) to specify the range of indexes for which you want to collect statistics. Access to the statistics is handled through the Virtual System Tables (VSTs). Index statistics are stored in the _IndexStat VST. To obtain index numbers, use the following 4GL code:

```
FOR EACH _file:
     DISPLAY _file._file-name.
     FOR EACH _index WHERE _index._file-recid = RECID(_file):
           DISPLAY _idx-num _index-name.
     END.
END.
```

This results in the following output:

```
File-Name
filename

_idx-num Index-Name
     n1   index name1
     n2   index name2
     n3   index name3
```

18.8.6 Base Table (-basetable)

Operating System and Syntax	UNIX Windows	-basetable *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	–	–	–	–

n

The starting table number in the range of tables for which you want to track access statistics.

Use Base Table (-basetable) with Table Range Size (-tablerangesize) to specify the range of tables for which you want to collect statistics. Access to the statistics is handled through the Virtual System Tables (VSTs). Table statistics are stored in the _TableStat VST. To obtain table numbers, use the following 4GL code:

```
FOR EACH _file:
     DISPLAY _file-num _file.
END.
```

This results in the following output:

```
_File-Number File-Name
     n1   table name1
     n2   table name2
     n3   table name3
```

18.8.7 Before-image Buffers (-bibufs)

Operating System and Syntax	UNIX Windows	-bibufs *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	–	3	–	5

n

The number of before-image buffers.

Use Before-image Buffers (-bibufs) to specify the number of before-image buffers. This parameter is useful only when running the before-image writer (BIW). The BIW continually writes the filled before-image buffers to disk, making the buffers available to other client and server processes. Without a BIW writing the buffers, any gain from increasing the number of buffers is negligible.

18.8.8 Threshold Stall (-bistall)

Operating System and Syntax	UNIX Windows	-bistall		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	–	–	–	–

Use Threshold Stall (-bistall) with Recovery Log Threshold (-bithold) to quiet the database when the recovery log threshold is reached, without performing an emergency shutdown. When you use -bistall, a message is added to the database log (.lg) file stating that the threshold stall is enabled.

18.8.9 Recovery Log Threshold (-bithold)

Operating System and Syntax	UNIX Windows	-bithold *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	System dependent[1]	System dependent[1]	–	–

[1] Limited by available disk space.

n

An integer specifying the threshold, in MB.

Use Recovery Log Threshold (-bithold) to set the maximum size to which recovery log files can grow. The recommended threshold is between 3% and 100% of the largest possible recovery log file size, rounded to the nearest cluster boundary. Once the threshold is reached, the database performs an emergency shutdown. Using the -bistall parameter with -bithold prevents the emergency shutdown; instead, the database ceases activity until the BI extent is emptied.

18.8.10 SQL-92 Server Java Classpath (-classpath)

Operating System and Syntax	UNIX Windows	-classpath *pathname*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	–	–	–	–

pathname

Specifies the pathname of the classpath.

Use SQL-92 Classpath (-classpath) to identify the Java classpath to use when starting an SQL server. SQL-92 database brokers use this parameter when launching the Java Virtual Machine (JVM) to execute stored procedures. The default is to use the current environment variable CLASSPATH setting. You do not use this parameter directly.

18.8.11 Conversion Map (-convmap)

Operating System and Syntax	UNIX Windows	-convmap *filename*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
CS, DBS	–	–	–	–

filename

The pathname of your CONVMAP file.

Use Conversion Map (-convmap) to identify the CONVMAP file to use for code page conversions, collation orders, and case conversions. By default, Progress uses the convmap.cp file in the DLC directory. You can create a CONVMAP file by using the PROUTIL utility with the CODEPAGE-COMPILER qualifier. See the *Progress Internationalization Guide* for more information on CONVMAP files.

18.8.12 Case Table (-cpcase)

Operating System and Syntax	UNIX Windows	-cpcase *tablename*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
CS, DBS, OID	–	–	Basic	Basic

tablename

The name of a case table in the convmap.cp file.

Use Case Table (-cpcase) to specify the case table. This table establishes case rules for the code page that Progress uses in memory. The code page is specified by the Internal Code Page (-cpinternal) parameter. The case rules are used by the CAPS and LC functions. Also, in a character field format you can use an exclamation point (!) to tell Progress to convert all characters to uppercase during input.

To retrieve the value of this startup parameter at run time, use the SESSION System handle.

18.8.13 Collation Table (-cpcoll)

Operating System and Syntax	UNIX Windows	-cpcoll *tablename*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
CS, DBS, OID	–	–	Basic	Basic

tablename

The name of a collation table within the convmap.cp file.

Use Collation Table (-cpcoll) to identify a collation table that Progress uses with the code page in memory. The code page is specified by the Internal Code Page (-cpinternal) parameter.

Progress uses the collation rules that you specify to compare characters and sort records if a BY clause cannot be satisfied by an index. The collation rules specified with the -cpcoll parameter take precedence over the collation specified for any database Progress accesses during the session, except when Progress uses or modifies pre-existing indexes. When you update or rebuild a database's indexes, Progress uses the collation rules originally defined for that database.

If you do not use -cpcoll, Progress uses the language collation rules defined for the first database on the command line. If you do not specify a database on the command line, Progress uses the collation rules with the name "basic."

To retrieve the value of this startup parameter at run time, use the SESSION System handle.

See the *Progress Internationalization Guide* for more information on collation tables.

18.8.14 Internal Code Page (-cpinternal)

Operating System and Syntax	UNIX Windows	-cpinternal *codepage*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
CS, DBS, DS, OID	–	–	iso8859-1[1]	iso8859-1[1]

[1] If you are using Progress/400, the default code page is ibm037.

codepage

The name of the code page that Progress uses in memory.

Use Internal Code Page (-cpinternal) to identify the code page that Progress uses in memory and for graphical clients. For graphical clients, the -cpinternal code page should be the same code page that the operating system uses. If you do not use -cpinternal, the iso8859-1 code page is used by default.

NOTE: Do not use a 7-bit table with -cpinternal. Use 7-bit tables for converting data from a 7-bit terminal to another code page only. Do not use them for character conversion in memory or for the database.

To retrieve the value of this startup parameter at run time, use the CPINTERNAL attribute of the SESSION System handle.

18.8.15 Log File Code Page (-cplog)

Operating System and Syntax	UNIX Windows	-cplog *codepage*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
CS, DBS, DS, OID	–	–	-cpstream	-cpstream

codepage

The name of the code page for messages written to the log file.

Use Log File Code Page (-cplog) to identify the code page that Progress uses to write messages to the log (.lg) file. If you do not specify a value, the default is the code page specified by Stream Code Page (-cpstream).

To retrieve value of this startup parameter at run time, use the CPLOG attribute of the SESSION System handle.

18.8.16 Print Code Page (-cpprint)

Operating System and Syntax	UNIX Windows	-cpprint *codepage*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
CS, DBS, DS, OID	–	–	-cpstream	-cpstream

codepage

The name of the code page used for printer output.

Use Print Code Page (-cpprint) to identify the code page Progress uses when it prints. When you print a file, the code page specified by -cpprint overrides the code page specified by Stream Code Page (-cpstream).

To retrieve the value of this startup parameter at runtime, use the SESSION System handle.

18.8.17 R-code In Code Page (-cprcodein)

Operating System and Syntax	UNIX Windows	-cprcodein *codepage*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
CS, DBS, DS, OID	–	–	-cpinternal	-cpinternal

codepage

The name of the code page for reading r-code text segments.

Use R-code In Code Page (-cprcodein) to read the r-code text segments, as if they were written in the code page specified by -cprcodein, and convert them to the Internal Code Page (-cpinternal) code page. Usually when Progress reads r-code, it converts text segments to the code page specified by Internal Code Page (-cpinternal).

CAUTION: This parameter is for use during very rare situations and in general should not be used. Progress reads text segments as if they are written in the code page specified by -cprcodein, even if the text segments were written with a different code page. For example, if you use the following startup parameters and run a .r file written with code page IBM850, Progress converts the text segments from ISO8859-1 to ibm861. This can produce incorrect results if the .r file was correctly labeled internally as IBM850:

```
-cprcodein ISO8859-1 -cpinternal ibm861
```

To retrieve the value of this startup parameter at run time, use the SESSION System handle. To determine the code page of an r-code file, use the RCODE-INFO handle.

18.8.18 Stream Code Page (-cpstream)

Operating System and Syntax	UNIX Windows	-cpstream *codepage*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
CS, DBS, DS, OID	–	–	ibm850	ibm850

codepage

The name of the code page for stream I/O.

Use Stream Code Page (-cpstream) to identify the code page Progress uses for stream I/O. Character terminals use the code page you specify for -cpstream unless you also specify a value for Terminal Code Page (-cpterm), Print Code Page (-cpprint), or Log File Code Page (-cplog).

Stream I/O consists of the following elements:

- Terminals (includes character terminals and DOS Protected mode, but does not include graphical interfaces or the Windows character interface)
- Data (.d) files
- READ-FILE, WRITE-FILE, and INSERT-FILE methods for the EDITOR widget
- INPUT FROM and OUTPUT TO statements
- All compilable files (.p, .w, .i, etc.)
- Compiler-generated LISTING, XREF, and PREPROCESS files

NOTE: Do not use a 7-bit table with -cpstream. Use 7-bit tables for converting data from a 7-bit terminal to another code page only. Do not use them for character conversion in memory or for the database.

To retrieve the value of this startup parameter at run time, use the SESSION System handle. To determine the code page of an r-code file, use the RCODE-INFO handle.

18.8.19 Terminal Code Page (-cpterm)

Operating System and Syntax	UNIX Windows	-cpterm *codepage*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
CS, DBS, DS, OID	–	–	-cpstream	-cpstream

codepage

The name of the code page for character terminals.

Use Terminal Code Page (-cpterm) to identify the code page of your character terminals. This parameter allows you to specify a different code page for character terminals than used by the rest of stream I/O, which is set by Stream Code Page (-cpstream).

NOTE: You can use a 7-bit table with -cpterm.

To retrieve the value of this startup parameter at run time, use the SESSION System handle. To determine the code page of an r-code file, use the RCODE-INFO handle.

18.8.20 Direct I/O (-directio)

Operating System and Syntax	UNIX Windows	-directio		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
CC, DBS	–	–	Not enabled	Not enabled

Use Direct I/O (-directio) to open all files in unbuffered mode, which enables Progress to use an I/O technique that bypasses the operating system buffer pool and transfers data directly from a buffer to disk. This technique has several advantages over buffered reads and writes such as avoiding the overhead of maintaining the operating system buffer pool and eliminating competition for operating system buffers between Progress programs and other programs. The operating system buffer-pool algorithms are designed for efficient sequential file access; the Progress buffer-pool algorithms are more efficient for access to a Progress database.

You might improve performance by using the direct I/O feature. To use direct I/O, use Blocks in Database Buffers (-B) to increase the size of the Progress buffer pool, since Progress I/O does not pass through the operating system buffer pool. Also, decrease the size of the operating system buffer pool to compensate for the additional memory allocated to the database.

NOTE: Use asynchronous page writers (APWs). They improve database performance by performing overhead operations in the background.

18.8.21 Event Level (-evtlevel)

Operating System and Syntax	NT	-evtlevel *value*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS, CS	–	–	Normal	Normal

value

Use Event Level (-evtlevel) to specify the level of information that Progress writes to the NT Application Event Log. Valid values include:

- None: No Progress events are written to the NT Event Log.
- Brief: Progress Error and Warning messages are written to the NT Event Log.
- Normal: Progress Error and Warning messages are written to the NT Event Log along with any Progress message that is normally written to the log file (.lg). This is the default.
- Full: Progress Error, Warning, and Informational messages are written to the NT Event Log along with any messages generated by the Message Statement.

For more information about Progress and the NT Event Log, see the *Progress Installation and Configuration Guide Version 9 for UNIX* and *Progress Installation and Configuration Guide Version 9 for Windows*.

18.8.22 Before-image Cluster Age (-G)

Operating System and Syntax	UNIX Windows	-G *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	32	0	60	60

n

The number of seconds the database engine waits.

Use Before-image Cluster Age (-G) to specify the number of seconds before the database engine reuses a before-image cluster.

18.8.23 Group Delay (-groupdelay)

Operating System and Syntax	UNIX Windows	-groupdelay *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	1,000	0	0	1

n

The number of milliseconds a transaction waits before committing.

Use Group Delay (-groupdelay) to increase performance when Delayed BI File Write (-Mf) is set to zero. When the Group Delay is set greater than zero (0), Progress uses a technique known as Group Commit. When using Group Commit, a transaction spools its end note to the BI buffer and waits a short time until the buffer becomes full and is written to disk, or waits for other transactions to end and store their end notes in the BI buffer so that several transactions are committed by the same synchronous write. In this manner, Group Commit benefits overall performance, although each individual transaction might take slightly longer.

18.8.24 Host Name (-H)

Operating System and Syntax	Windows (SNA)	-H PLU=*plu-name*		
	UNIX Windows (TCP)	-H { *host-name* | localhost[1] }		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
CC, DBS, DS	–	–	–	–

[1] `localhost` does not apply to DataServers.

plu-name

On Windows systems running the Progress/400 DataServer, specifies the Partner LU name. The default for Windows 95 is the AS/400 host name from the `CONFIG.PCS file`.

host-name

The name (address) of the database server machine. This name is assigned to the machine in your TCP/IP hosts file.

`localhost`

A reserved word that specifies that the Database server communicates only with clients on the database server machine.

Use Host Name (`-H`) to identify the host name.

NOTE: This parameter has a special purpose when used with the Progress/400 DataServer and SNA. If you are connecting to multiple AS/400 machines, you must supply this parameter for each AS/400.

18.8.25 Hash Table Entries (-hash)

Operating System and Syntax	UNIX Windows	-hash *n*	
Use With	**Maximum Value**	**Minimum Value**	**Default**
CC, DBS	–	13	Approximately 1/4 of the -B value

n

The number of hash table entries to use for the buffer pool.

CAUTION: Do not use this parameter unless directed to do so by Progress Software Technical Support.

18.8.26 No Crash Protection (-i)

Operating System and Syntax	UNIX Windows	-i	
Use With	**Maximum Value**	**Minimum Value**	**Default**
CC, DBS	–	-	-

Use No Crash Protection (-i) to run the database without integrity or database recovery. When running without database integrity, the database engine writes fewer data and before-image blocks to the disk. In this mode, some procedures (such as those that create and delete large numbers of records) may run significantly faster than if they are running with database integrity.

When running with the -i parameter, transaction undo is supported. Therefore, there will still be a before-image file, which might grow quite large during very long transactions.

Use this parameter to do bulk data loading or for large batch runs. It reduces the number of disk input or output operations. Loading a database for the first time is a good example of a use for this parameter.

CAUTION: If you run your database with the -i parameter and the database fails for any reason, you cannot recover the database.

Do not use the -i parameter unless you have a complete backup of the database and can rerun procedures in case of a system failure. If the system fails during a Progress session that started without crash protection, restore the backup copy and rerun the necessary procedures. For information on restoring a database, see Chapter 8, “Recovering a Database.”

18.8.27 Index Range Size (-indexrangesize)

Operating System and Syntax	UNIX Windows	-indexrangesize *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	–	–	–	–

n

The number of indexes for which you want to track access statistics.

Use Index Range Size (-indexrangesize) to specify the number of indexes for which you want to collect statistics from virtual system tables (VSTs). See Chapter 20, "Virtual System Tables,"for more information on VSTs.

18.8.28 Lock Table Entries (-L)

Operating System and Syntax	UNIX Windows	-L *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	System dependent[1]	32	–	8192

[1] Limited by available memory.

n

The number of entries in the record locking table. If you specify a value that is not a multiple of 32, Progress rounds the value you specify to the next highest multiple of 32.

Use Lock Table Entries (-L) to change the limits of the record locking table. Each record that is accessed and locked by a user takes one entry. This is true whether the record is accessed with SHARE-LOCK or EXCLUSIVE-LOCK. Increase the size of the lock table if the following message appears:

```
SYSTEM ERROR: Record lock table too small. Increase -L parameter.
```

This message might also indicate that a particular procedure should be restructured into smaller transactions or should be run in single-user rather than multi-user mode. When lock table limits are exceeded, check to make sure transactions are not too large before increasing the lock table size.

If a user process tries to acquire a lock and the lock table overflows, the user's program is aborted, but the server continues to operate. Any partial transactions are undone.

Note that two record locks are acquired when records are accessed with the BREAK BY option (in DO, FOR EACH, or REPEAT statements).

Each lock table entry takes 18 bytes on typical systems.

18.8.29 Auto Server (-m1)

Operating System and Syntax	UNIX Windows	-m1		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	–	–	–	–

Use Auto Server (-m1) to start an auto server. The Progress broker uses the auto server internally to start a remote user server. This is the default. You will never have to use this parameter directly.

18.8.30 Manual Server (-m2)

Operating System and Syntax	UNIX Windows	-m2		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	–	–	–	–

Use Manual Server (-m2) to manually start a remote user server after you start a broker (servers are generally started automatically by the broker process). Use this parameter in the following cases:

- For debugging purposes, to start servers directly and observe their behavior
- On systems where automatic server generation is not possible

18.8.31 Secondary Login Broker (-m3)

Operating System and Syntax	UNIX Windows	-m3		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	–	–	–	–

In a network environment where more than one broker is using the same protocol, use Secondary Login Broker (-m3) to start each secondary broker. The secondary broker logs in clients and starts remote user servers.

18.8.32 Maximum Clients Per Server (-Ma)

Operating System and Syntax	UNIX Windows	-Ma *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	2048	1	–	5 users/server

n

The maximum number of remote users per database server. The default is the Maximum Number of Users (-n) parameter value, divided by the Maximum Number of Servers (-Mn) parameter value.

Use Maximum Clients per Server (-Ma) to specify the maximum number of remote users per database server. The Maximum Clients per Server (-Ma), Minimum Clients per Server (-Mi), and Maximum Servers (-Mn) startup parameters apply only to databases that are accessed from remote network nodes.

In most cases, the default behavior is desirable. Note that the default calculation is usually high because it assumes that all users are remote users, while the number specified with -n includes local users. If servers become overloaded with clients, reset the -Mn parameter to increase the number of servers.

NOTE: For SQL-92 database brokers, there is a limit of one client per server. The maximum value does not apply.

18.8.33 Maximum Dynamic Server (-maxport)

Operating System and Syntax	UNIX Windows	-maxport *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	System dependent	System dependent	–	2000

n

The port number that is the highest in a specified range.

Use Maximum Dynamic Server (-maxport) to specify the highest port number in a specified range of port numbers accessible to a client. You specify the lowest port number with the -minport parameter. The range of port numbers defined by the -maxport and -minport parameters provides client access to a Progress server that is behind a firewall. Some operating systems choose transient client ports in the 32,768-to-65,535 range. Choosing a port in this range might produce unwanted results.

18.8.34 Delayed BI File Write (-Mf)

Operating System and Syntax	UNIX NT	-Mf *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	32,768	0	0[1]	3

[1] Default is 3 for batch jobs.

n

Value in seconds of the delay before the database engine synchronously writes to disk the last before-image (BI) file records at the end of each transactions. It also specifies the interval that the broker process wakes up to make sure all BI file changes have been written to disk. The default is 3 for single-user batch jobs and for multi-user databases using shared memory. Otherwise, the default is 0.

Use Delayed BI File Write (-Mf) to improve performance on a heavily loaded system. Using -Mf does not reduce database integrity. However, if there is a system failure, it is possible the last few completed transactions will be lost (never actually written to the BI file).

When running with full integrity, at the end of each transaction the database engine does a synchronous write to disk of the last BI file block. This write guarantees that the completed transaction is recorded permanently in the database. If the user is notified that the transaction has completed and the system or database manager crashes shortly afterwards, the transaction is not lost.

Do not set -Mf on a lightly loaded system with little database update activity. Under these conditions, the extra BI write is very important and does not impact performance. On a heavily loaded system, however, the BI write is less important (the BI block will be written to disk very soon anyway), and has a significant performance penalty. Setting -Mf to delay this extra BI write saves one write operation per transaction, which can significantly improve performance. The extra BI file write is delayed by default for batch jobs.

The last BI file record is only guaranteed to be written out to disk when a user logs out, or when the server or broker process terminates normally. On multi-user systems, the *n* argument determines the oldest completed transaction that can be lost.

18.8.35 Minimum Clients Per Server (-Mi)

Operating System and Syntax	UNIX NT	-Mi *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	–	1	–	1

n

The number of remote users on a server before the broker starts another server. See the "Maximum Servers (-Mn)" section.

Use Minimum Clients per Server (-Mi) to specify the number of remote users on a server before the broker starts another server (up to the maximum number of servers). In addition, -Mi and -Mn apply only to databases that are accessed from remote network nodes.

As remote users enter the database, the broker process starts just one server process for each *n* remote users, until the maximum number of servers (specified by the -Mn parameter) is started. If you specify a value of 1, the broker starts a new server for each of the first -Mn remote users. Subsequent remote users are distributed evenly among the servers until the maximum number of users (-n) or maximum clients per server (-Ma) limits are reached.

NOTE: If you are using SQL-92, you do not need to use -Mi because the limit is always one client per server.

Typically, you can leave -Mi and -Mn at their default values. If you significantly increase -Mn, you should also increase -Mi. For example, if you set -Mn to 10 to accommodate up to 40 or more remote users, increase -Mi to 3 or 4 to prevent a situation where 10 servers were started for just 10 remote users.

18.8.36 Minimum Dynamic Server (-minport)

Operating System and Syntax	UNIX NT	-minport *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	–	1,025	–	1,025

n

The port number that is the lower in a specified range.

Use Minimum Dynamic Server (-minport) to specify the lowest port number in a specified range of port numbers accessible to a client. You specify the higher port number with the -maxport parameter. Ports below 1025 are usually reserved for system TCP and UDP. The range of port numbers defined by the -maxport and -minport parameters provides client access to a Progress server that is behind a firewall. This communication is possible only when the access to the server can be limited.

18.8.37 Maximum Servers (-Mn)

Operating System and Syntax	UNIX NT	-Mn *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	512	1	–	4

n

The maximum number of remote client servers that can be started on the system.

Use Maximum Servers (-Mn) to limit the number of remote user servers that can be started by the broker process. The performance tradeoff to consider is swapping overhead for many servers versus overloading (slowing down) a server with too many clients.

This parameter applies only to databases that are accessed from remote network nodes.

Also, use Minimum Clients per Server (-Mi) to adjust the actual number of servers in use. See the Maximum Clients Per Server (-Ma) and Minimum Clients per Server (-Mi) startup parameters for more information.

18.8.38 Servers Per Protocol (-Mp)

Operating System and Syntax	UNIX NT	-Mp *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	Value of -Mn	1	–	value of -Mn

n

The number of servers a broker can start.

Use Servers Per Protocol (-Mp) with Secondary Login Broker (-m3) in database networks that use more than one network protocol. This parameter limits the number of servers that the broker can start to serve remote users for any one protocol. The total number of servers for all protocols is still limited by the Maximum Servers (-Mn) parameter.

18.8.39 Maximum Servers Per Broker (-Mpb)

Operating System and Syntax	UNIX NT	-Mpb *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	–	–	–	–

n

The number of servers each broker can start.

Use Maximum Server Per Broker (-Mpb) to specify the maximum number of servers that multiple brokers can start to serve remote users for any one protocol.

18.8.40 VLM Page Table Entry Optimization (-Mpte)

Operating System and Syntax	Digital UNIX	-Mpte		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	–	–	–	–

Use VLM Page Table Entry Optimization (-Mpte) to allocate shared memory in multiples of 8 MB at server startup for VLM64 support. This function is a binary switch that is off by default. The -Mpte startup parameter turns the function on.

18.8.41 Shared-memory Overflow Size (-Mxs)

Operating System and Syntax	UNIX NT	-Mxs *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	Varies[1]	1	–	4KB + (*n* * 300)

[1] The maximum is limited only by the size of the signed integer data type on the system.

n

The size of the shared-memory overflow area in kilobytes.

Use Shared-memory Overflow (-Mxs) to replace the default value of the shared-memory overflow area; it does not increase it. The overflow area is appended to the shared-memory area. If the overflow area is too small, the database engine exits with the following message:

```
SYSTEM ERROR: Out of free shared memory. Use -Mxs to increase.
```

Depending on the operating system, the database engine rounds the shared-memory area size to the next 512-byte or 4K boundary.

18.8.42 Network Type (-N)

Operating System and Syntax	UNIX Windows	-N *network-type*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
CC, DBS	–	–	–	System dependent

network-type

The network communications protocol.

Use Network Type (-N) to specify the network communications protocol supported by Progress: TCP or SNA (Progress/400). SNA allows Windows and certain UNIX clients to access Progress/400 servers. For more information, see the *Progress/400 Product Guide*.

18.8.43 Number Of Users (-n)

Operating System and Syntax	UNIX Windows	-n *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	10,000	–	–	20

n

The maximum number of Progress users on the system. After *n* users have connected to the Progress database, additional user startup attempts are rejected.

-n must be high enough to include local and remote users as well as background writers (APWs, BIWs, and AIWs), PROWDOG processes, and PROMON sessions. For more information, see the Maximum Clients Per Server (-Ma) and Maximum Servers (-Mn) startup parameters.

18.8.44 Parameter File (-pf)

Operating System and Syntax	UNIX Windows	-pf *filename*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
CC, CS, DBS, DS, OID	–	–	–	–

filename

The name of a parameter file that contains startup parameters to run Progress.

Use Parameter File (-pf) to name a parameter file that includes any number of startup parameters to run Progress. This parameter is especially useful if you regularly use the same parameters to start Progress, or if more parameters are specified than can fit on the command line. This parameter can be included within the parameter file itself to reference another parameter file.

Use multiple instances of -pf to name multiple parameter files. This allows you to specify application-specific parameters in one parameter file, database-specific parameters in a second parameter file, and user-specific parameters in yet another file.

18.8.45 Pin Shared Memory (-pinshm)

Operating System and Syntax	UNIX Windows	-pinshm		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	–	–	–	–

The Pin Shared Memory (-pinshm) parameter does not have any arguments.

Using -pinshm will prevent the database engine from swapping shared memory contents to disk, which can help improve performance.

18.8.46 Configuration Properties File (-properties)

Operating System and Syntax	UNIX Windows	-properties *filename*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	–	–	–	–

filename

The name of the properties file.

Use Configuration Properties File (-properties) to identify the properties file that the AdminServer uses internally to specify startup parameters when starting a database server or servergroup. The default is $DLC/properties/conmgr.properties. You do not use this parameter directly.

18.8.47 Buffered I/O (-r)

Operating System and Syntax	UNIX Windows	-r		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
CC	–	–	Unbuffered I/O	Unbuffered I/O

Use Buffered I/O (-r) to enable buffered I/O to the before-image file. In most cases, avoid using this parameter because it might put database integrity at risk.

CAUTION: A database running with the -r parameter cannot be recovered after a system failure. If the system fails, you must restore the database from a backup and restart processing from the beginning.

18.8.48 Service Name (-S)

Operating System and Syntax	UNIX Windows	-S { *service-name* | *port-number* }		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
CC, DBS	–	–	–	–

service-name

The name of the service to be used by the broker process

port-number

The port number of the host; if using Progress Explorer, the port number of the NameServer

Use Service Name (-S) to specify the service or port number to be used when connecting to a broker process or used by a broker process on the host machine. You must use this parameter when you are starting:

- A broker or server on a machine that will serve remote users
- A multi-user session as a remote user

The system administrator must make an entry in the `services` file that specifies the server or broker name and port number.

When the broker spawns a server, the server inherits all of the network parameters (except the Service Name parameter) from the broker. Because there is no restriction on the number of brokers you can start, you can have multiple brokers running with different network parameters. See also the Server Group (`-servergroup`) startup parameter description.

Table 18–7 shows how the broker, server, and remote client interpret each of their parameters when you use the -S parameter.

Table 18–7: Parameter Interpretation With Service Name (-S)

Module	Interpretation
Broker	Parameters apply to the connections on which the broker is listening for connection requests from remote clients.
Server	Parameters apply to the connection between the server and the remote client.
Remote Client	Parameters identify the connection parameters to the broker or the server.

To run multi-user Progress from a remote network node, use both the Host Name (-H) and Service Name (-S) parameters.

18.8.49 Semaphore Sets (-semsets)

Operating System and Syntax	UNIX ONLY	-semsets *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	Maximum Number of Users + 1	1	–	1

n

An integer specifying the number of semaphore sets available to the broker.

Use Semaphore Sets (-semsets) to change the number of semaphore sets available to the broker. When more than 1,000 users connect to a single database, there might be high contention for the semaphore set. If there is a lot of semaphore contention on a system, using multiple semaphore sets helps improve performance on high user counts.

18.8.50 Server Group (-servergroup)

Operating System and Syntax	UNIX Windows	-servergroup *name*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	–	–	–	–

name

Specifies the name of the logical collection of server processes.

Use Server Group (-servergroup) to identify the logical collection of server processes to start. The *name* you specify must match the name of a servergroup in the conmgr.properties file. Use the Progress Explorer tool to create servergroups and save them in the conmgr.properties file. Do not edit the conmgr.properties file directly. To start a database configuration use Progress Explorer or the DBMAN utility.

18.8.51 Spin Lock Retries (-spin)

Operating System and Syntax	UNIX NT	-spin *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	–	0	–	0

n

The number of times a process tries to acquire a latch before pausing.

Use Spin Lock Retries (-spin) to set a value to use the spin lock mechanism or a system of semaphores and queues. If the value of *n* is greater than 0, a spin lock algorithm is used for shared-memory data structures. When a process has to lock a shared-memory structure, the process tries up to *n* times to acquire the latch for that structure. If it has not acquired the latch in *n* tries, then the process pauses, or naps. The length of the pause increases gradually if the process repeatedly fails to acquire a latch. After the allotted nap time, the process wakes up and tries to acquire the lock again. If it fails to acquire the lock, it will retry up to the number of tries specified by *n*.

If the value of *n* is 0, a system of semaphores and queues is used instead of spin locks. The spin lock algorithm is much more efficient than using semaphores when you have multiple processors.

In some cases, if the value of *n* is 1, it might improve performance even on a single-processor machine because Progress uses the spin lock mechanism. Do not set this parameter to a value larger than 1 if your machine has only one processor.

On multi-processor machines, try a value of 2,000. If this causes too much CPU usage, reduce the value. If you have many fast processors, a value as high as 10,000 might be effective.

You can evaluate the `-spin` values through the Progress Monitor (PROMON utility) R&D options. See Appendix A, "Progress Monitor R&D Options," of this manual. The `-spin` parameter controls the performance indicator called resource waits. By setting the `-spin` value higher, you can reduce the resource waits. Note that when setting the `-spin` value higher ceases to reduce the resource waits, continuing to set it higher can adversely effect CPU utilization. To view the resource waits value:

- Access PROMON and enter R&D at the main menu.
- Choose option 3, Other Displays, then choose option 1, Performance Indicators, to view the resource waits. Resource waits is the last item reported in the listing.

18.8.52 Table Range Size (-tablerangesize)

Operating System and Syntax	UNIX Windows	`-tablerangesize` *n*		
Use With	**Maximum Value**	**Minimum Value**	**Single-user Default**	**Multi-user Default**
DBS	–	–	–	–

n

The number of tables for which you want to track access statistics.

Use Table Range Size (`-tablerangesize`) to specify the number of tables for which you want to collect statistics.

19

Database Administration Utilities

This chapter describes the Progress database administration utilities, in alphabetical order. It discusses the purpose, syntax, and primary parameters for each operating system.

Figure 19–1 shows the conventions used in utility command syntax.

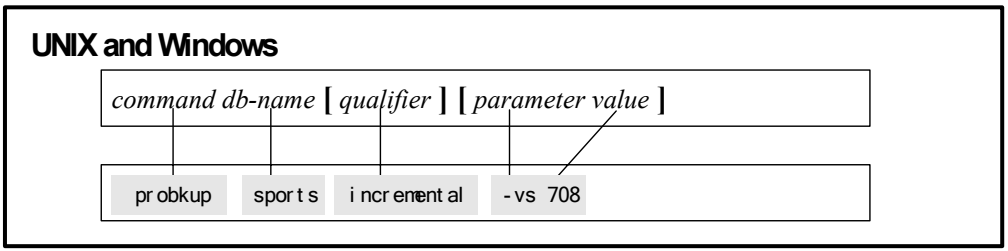


Figure 19–1: Utility Syntax Conventions

Table 19–1 describes each of the command components.

Table 19–1: Progress Command Components

Component	Description
command	Progress executable
qualifier	Additional command specification
parameter	Operating criteria for the command
value	Numeric value or file specification

NOTE: Enter parameters on UNIX and Windows exactly as shown in the syntax descriptions.

DBMAN Utility

Starts, stops, or queries a database. Before you can use the DBMAN command-line utility, you must use the Progress Explorer Database Configuration Tool to create the database configuration and store it in the `conmgr.properties` file.

SYNTAX

Operating System	Syntax
UNIX Windows	`dbman [ -host host-name -port port-number | service-name -user user-name ] -database db-name [-config config-name -start|-stop|-query]`

`-database` *db-name*

Specifies the name of the database you want to start. It must match the name of a database in the `conmgr.properties` file.

`-config` *config-name*

Specifies the name of the configuration with which you want to start the database.

`-start`

Starts the database *db-name* as defined by the configuration *config-name*.

`-stop`

Stops the database *db-name*.

`-query`

Queries the Connection Manager for the status of the database *db-name*.

`-host` *host-name*

Identifies the host machine where the AdminServer is running. The default is the local host. If your AdminServer is running on a remote host, you must use the `-host` *host-name* parameter to identify the host where the remote AdminServer is running.

`-port` *port-number*|*service-name*

Identifies the port that the AdminServer is listening on. If your AdminServer is running on a remote host, you must use the `-port` *port-number* parameter to identify the port on which the remote AdminServer is listening. The default port number is 20931.

`-user` *user-name*

If your AdminServer is running on a remote host, you must use the `-user` *user-name* parameter to supply a valid user name for that host. You will be prompted for the password.

NOTES

- When you specify a user name with the `-user` parameter, Windows supports three different formats:
  - A user name as a simple text string, such as "`mary`," implies a local user whose user account is defined on the local server machine, which is the same machine that runs the AdminServer.
  - A user name as an explicit local user name, in which the user account is defined on the same machine that runs the AdminServer, except the user name explicitly references the local machine domain, for example "`\mary`".
  - A user name as a user account on a specific NT domain. The general format is *Domain*`\`*User*, in which the *User* is a valid user account defined within the domain and the *Domain* is any valid NT server, including the one where the AdminServer is running.
- Do not edit the `conmgr.properies` file directly. Instead, use the Progress Explorer Database Configuration Tool.
- DBMAN supports the use of internationalization startup parameters such as, `-cpinternal` *codepage* and `-cpstream` *codepage*. See Chapter 18, "Database Startup Parameters," for a description of each database-related internationalization startup parameter.

- The `conmgr.properties` file stores the database, configuration, and servergroup properties. For example:

(1 of 2)

```
# Connection Manager Properties File
#
%% version 1.0
#
# The following are optional configuration properties and their default
# values. The legacy option, if applicable, is listed after the second
# comment. Property values set at this level become the default values for
# all configuration subgroups.
#
[configuration]
#   afterimagebuffers=5                 # -aibufs
#   afterimagestall=true                # -aistall
#   beforeimagebufferedwrites=false     # -r
#   beforeimagebuffers=5                # -bibufs
#   beforeimageclusterage=60            # -G
#   beforeimagedelaywrites=3            # -Mf
#   blocksindatabasebuffers=0           # -B (calculated s 8*(-n))
#   logcharacterset=iso8859-1           #
#   collationtable=basic                # -cpcoll
#   crashprotection=true                # -i
#   databasecodepage=basic              # -cpdb
#   directio=false                      # -directio
#   hashtableentries=0                  # -hash (calculated as (-B)/4)
#   locktableentries=10000              # -L
#   maxservers=4                        # -Mn
#   maxusers=20                         # -n
#   nap=1                               # -nap
#   napmax=1                            # -napmax
#   pagewritermaxbuffers=25             # -pwwmax
#   pagewriterqueuedelay=100            # -pwqdelay
#   pagewriterqueuemin=1                # -pwqmin
#   pagewriterscan=1                    # -pwscan
#   pagewriterscandelay=1               # -pwsdelay
#   semaphoresets=1                     # -semsets
#   sharedmemoryoverflowsize=0          # -Mxs
#   spinlockretries=0                   # -spin
#   sqlyearoffset=1950                  # -yy

[configuration.sports2000.default]
    database=sports2000
    displayname=default
    servergroups=sports2000.default.default4gl,
                 sports2000.default.defaultsql
```

(2 of 2)

```
# The following are optional database properties and their default
# values. Property values set at this level become the default values for
# all database subgroups.
#
[database]
#   autostart=false              # autostart the defaultconfiguration
#   databasename=mydemo          # absolute or relative path + database name
#defaultconfiguration=defaultdb.defaultconfig

[database.sports2000]
    autostart=true
    configurations=sports2000.default
    databasename=[[work-dir]]\sports2000
    defaultconfiguration=sports2000.default
    displayname=Sports2000

[environment]
# The following are optional server group properties and their default
# values. The legacy option, if applicable, is listed after the second
# comment. Property values set at this level become the default values for
# all servergroup subgroups.
#
[servergroup]
#   host=localhost                  # -H
#   initialserver=0
#   maxclientsperserver=0           # -Ma (calculated value)
#   maxdynamicport=0                # -maxport (5000 for NT;2000 for UNIX)
#   messagebuffersize=350           # -Mm (4gl only)
#   minclientperserver=1            # -Mi
#   mindynamicport=3000             # -miniport (3000 - NT; 2000 - UNIX)
#   networkclientsupport=true       # false for self-service
#   port=0                          # -S ; Must be non-zero
#                                   # when networkclientsupport=true
#reportinginterval=1                # -rpint (4gl only)
#serverexe=<4gl server location>    # -mprosrv (4gl only)
#servicename=                       # -
#type=4gl                           # -

[servergroup.sports2000.default.default4gl]
    configuration=sports2000.default
    displayname=default4gl
    port=2500
    type=4gl

[servergroup.sports2000.default.defaultsql]
    configuration=sports2000.default
    displayname=defaultsql
    port=2500
    type=sql
```

_DBUTIL CMPDB Qualifier

Allows you to compare two databases.

SYNTAX

Operating System	Syntax
UNIX Windows	`_dbutil cmpdb` *db-name1 db-name2* **[**`-t`**|**`-1`**|**`-12`**]**

db-name1

Name of the first database to be compared.

db-name2

Name of the second database to be compared.

`-t`

Tight compare. Checks all fields including backup counters and master block fields.

`-1`

Loose compare. Does not report differences in the master block last modification date or the back up update counter.

`-12`

Second loose compare. Does not report differences in the incremental backup field of the database block headers.

NOTE

- Note that in Version 9, CMPDB is no longer an executable. It must be run as a qualifier to _DBUTIL.

PROADSV Utility

Starts, stops, or queries the current installation of an AdminServer on UNIX.

SYNTAX

Operating System	Syntax
UNIX	proadsv { -start | -stop | -query } [-port *port-number*] [-adminport *port-number*] [-help]

-start

Starts the AdminServer.

-stop

Stops the AdminServer.

-query

Displays AdminServer status.

-port *port-number*

Specifies the listening port number for online command utilities, such as DBMAN. If a port number is not specified, it defaults to 20931.

-adminport *port-number*

Specifies the listening port number for communication between a servergroup and an AdminServer. The default port-number is 7832.

-help

Displays command-line help.

NOTES

- On Windows, the AdminServer runs as a service. The AdminServer is configured to automatically start. You can change the listening port for the AdminServer by adding `-port` *port-number* or `-adminport` *port-number* (either one or both) to the following registry keys:

```
HKEY_LOCAL_MACHINE\SOFTWARE\PSC\AdminService\version\StartupCmd

or

HKEY_LOCAL_MACHINE\SOFTWARE\PSC\AdminService\version\ShutdownCmd
```

  To change the default port, add `-port` or `-adminport` and the port number to the end of the value. If you add both `-port` and `-adminport`, be sure not to use the same port number. Be sure to leave a space between `-port` or `-adminport` and the port number. For example:

```
....AdminServer -port 9999 -adminport 7832
```

- To run more than one AdminServer on a single system, specify a unique `-adminport` *port-number* and `-port` *port-number* for each AdminServer. Failure to do so can result in communication errors between AdminServer and servergroups.

- The `-adminport` parameter was not available until Version 9.0B. To run both a Version 9.0A and Version 9.0B AdminServer on a single system, specify an `-adminport` port number to establish communication with the Version 9.0B AdminServer.

- On Windows 98, you can use the PROADSV utility with the `-query` parameter only.

- An AdminServer is installed on every system where you install a Progress database, ODBC DataServer or ORACLE DataServer, AppServer, NameServer, WebSpeed Transaction Server, or WebSpeed Messenger. The AdminServer must be running in order to use any of the Progress Explorer configuration tools or command-line configuration utilities, such as DBMAN.

- See Chapter 5, "Starting Up and Shutting Down," for additional information describing the Progress Explorer Framework, including an AdminServer.

- PROADSV supports the use of internationalization startup parameters such as `-cpinternal codepage` and `-cpstream` *codepage*. See Chapter 18, "Database Startup Parameters," for a description of each database-related internationalization startup parameter.

PROBKUP Utility

Backs up a Progress database, including the database, before-image files, and transaction log (TL) extents.

SYNTAX

Operating System	Syntax
UNIX Windows	probkup [online] *db-name* [incremental] *device-name* [-estimate | -vs *n* | -bf *n* | -verbose | -scan | -io *i* | -com | -red *i* | -no recover] ...

online

Indicates the backup is an online backup.

db-name

Specifies the database you want to back up.

incremental

Indicates the backup is an incremental backup.

device-name

Identifies a special device (for example, a tape drive) or a standard file. If *device-name* identifies a special device, Progress assumes the device has removable media, such as a tape or a floppy diskette. For Windows NT, use \\.\tape0 for the device name if you are backing up to a tape drive.

`-estimate`

Indicates that the backup will give a media estimate only. Use the Scan parameter when using the Incremental or Compression parameters to get an accurate estimate.

Use `-estimate` for offline backups only.

`-vs` *n*

Indicates the volume size in database blocks that can be written to each removable volume. Before PROBKUP writes each volume, it displays a message that tells you to prepare the next volume. After writing each volume, a message tells you to remove the volume.

If you use the Volume Size parameter, the value must be greater than the value of the Blocking Factor parameter.

If you do not use the Volume Size parameter, PROBKUP assumes there is no limit and writes the backup until completion or until the volume is full. When the volume is full, PROBKUP prompts you for the next volume.

`-bf` *n*

Indicates the blocking factor for blocking data output to the backup device. The blocking factor specifies how many blocks of data are buffered before being transferred to the backup device. NT uses a variable block size up to 4K. For all other operating systems, each block is the size of one disk block (1K on UNIX). The primary use for this parameter is to improve the transfer speed to tape-backup devices by specifying that the data is transferred in amounts optimal for the particular backup device. The default for the blocking factor parameter is 34.

`-verbose`

Directs the PROBKUP utility to display information during the backup. If you specify the Verbose parameter, PROBKUP displays “Backed up *n* blocks in `hh:mm:ss`” every 10 seconds. If you do not specify the Verbose parameter, the message appears only once when the backup is complete.

`-scan`

Directs PROBKUP to perform an initial scan of the database and to display the number of blocks that will be backed up and the amount of media the backup requires. You cannot use the Scan parameter for online backups.

For full backups, if you specify the Scan parameter as well as the Compression parameter, PROBKUP scans the database and computes the backup media requirements after the data is compressed.

`-io` *i*

Specifies an incremental overlap factor. The *incremental overlap factor* determines the redundancy among incremental backups. An incremental overlap factor of 1 on every backup allows for the loss of one incremental backup in a backup series, as long as the immediate predecessor of that backup is not also lost. An overlap factor of 2 allows for losing the two immediate predecessors. The default overlap factor is 0.

`-com`

Indicates that the data should be compressed prior to writing it on the backup media. The unused portion of index and record blocks is compressed to a 3-byte compression string. Free blocks are compressed to the length of their header, 16 bytes.

`-red` *i*

Sets the amount of redundancy to add to the backup for error correction. The value *i* is a positive integer that indicates the number of blocks for every error correction block. Progress creates an error correction block for every *i* blocks and writes it to the backup media. You can use error correction blocks to recover corrupted backup blocks. See Chapter 7, “Backing Up a Database,” for more information about error correction blocks and data recovery.

The lower the redundancy factor, the more error correction blocks Progress creates. If you specify a redundancy of 1, you completely duplicate the backup, block for block. Because of the amount of time and media required to create the error correction blocks, use this parameter only if your backup media is unreliable. The default for the redundancy parameter is 0 (no redundancy).

`-no recover`

Do not perform crash recovery before backing up the database, but back up the BI files.

NOTES

- The PROBKUP utility has been enhanced to use only Version 9 databases. There is no ability to read or write backup tapes from previous versions of the Progress database. When restoring a backup, the target database must contain the same physical structure as the backup version. For example, it must have the same number of storage areas, records, blocks, and blocksize.
- The minimum backup volume size is 34K.
- You cannot perform an online backup on:
  - A system running in single-user mode
  - A system without shared memory
  - A database that was started with the No Shared Memory (`-noshm`) parameter
- If you run the PROBKUP utility at the same time another process is accessing the same backup device, you might receive a sharing violation error.
- Before performing an online backup, Progress automatically switches to a new after-image (AI) file and establishes a reference point from which to start your roll-forward recovery. Before you perform an online backup, verify that the next AI file is empty. If the next file is not empty, Progress aborts the backup. See Chapter 11, "After-Imaging," for more information about AI files.
- If you use the Compression parameter, you reduce the size of your backup by 10 percent to 40 percent, depending on your database.
- If the BI file is not truncated before you perform a backup, Progress performs database recovery.
- See Chapter 7, "Backing Up a Database," for more information about performing backups.
- PROBKUP supports the use of internationalization startup parameters such as `-cpinternal` *codepage* and `-cpstream` *codepage*. See Chapter 18, "Database Startup Parameters," for a description of each database-related internationalization startup parameter.

ProControl Utility

A tool with a graphical user interface supplied with the Progress NT Server. The Progress NT Server permits Progress processes to run as registered NT services.

ProControl lets you perform the following tasks on a local or remote NT host:

- Start and stop ProService
- Start and stop Progress databases
- Start and stop Progress DataServer processes
- Set Progress environment variables
- Start Progress batch processes

NOTE: Using ProControl to perform these tasks ensures that Progress processes run as registered NT services. All of these tasks result in changes to the registry.

SYNTAX

Operating System	Syntax
Windows	`proctrl`

From the Control Panel or the Progress Program Group, double-click the ProControl icon. The ProControl icon looks like this:



When you start the utility the main ProControl window appears, as shown in Figure 19–2.

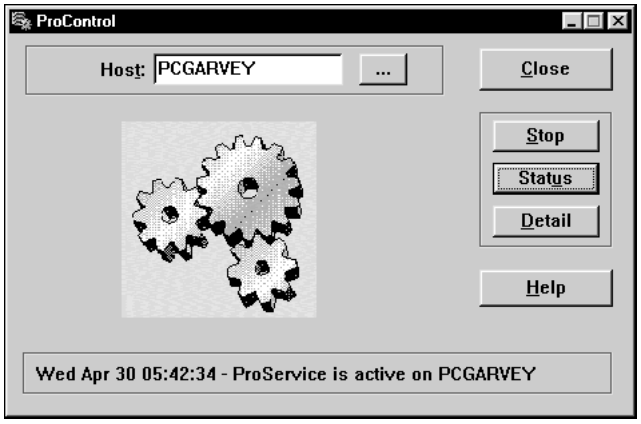


Figure 19–2: ProControl Main Window

NOTES

- The Progress NT Server stores server-specific environment information in the NT registry. Each time you use ProControl to perform an Insert, Copy, or Remove operation within one of the ProControl folders, it updates the appropriate subkey of the PSC subkey of the registry key HKEY_LOCAL_MACHINE. Note that when you use ProControl to perform a remove operation, ProControl removes the registry entries, but does not delete the files from the system:

```
HKEY_LOCAL_MACHINE
 SOFTWARE
  PSC
   ProService
    Version-Number
     Databases
     DataServers
     Environment
     OtherTasks
     ProControl
```

- When you insert a database entry using the Databases folder, ProControl adds the name of the database (and other information) to the Databases subkey of the PSC subkey of the HKEY_LOCAL_MACHINE key. If you then set an environment variable using the Databases Environment folder, ProControl adds that environment variable to the Databases subkey.

- If you do not use ProControl to insert Databases, DataServers, Environment variables, and Other Tasks, the registry is never updated and these processes do not run as NT services. Similarly, if you use some other means to update the registry, Progress cannot guarantee that the product will function properly.

- You manage access to the ProControl utility through the ProControl Access Group field (available through the ProControl Options folder). By default at installation, the ProControl Access Group field is set to the NT group called Everyone.

- You can specify access rights to shared memory for specific databases, DataServers, and tasks by setting the ACCESS environment variable within the Environment folder for that specific object, or globally for the ProService environment through the ProControl Environment folder.

PROCOPY Utility

Copies an existing database.

PROCOPY supports storage areas. Therefore, if a target database exists, it must contain at a minimum the same type and number of storage areas and same extent types as the source database. However, the number of extents in the storage areas of the target database do not need to match the number of extents in the source database. Progress will extend the existing extents in the target database to accommodate the possible increase in size.

If a target database does not exist, PROCOPY creates one using an existing ST file in the target database directory. If an ST file does not exist, PROCOPY creates the target database using the structure of the source database and places all of the extents in the same directory as the target database DB file, even when the source database resides in multiple directories.

SYNTAX

Operating System	Syntax
UNIX Windows	`procopy` *`source-db-name target-db-name`* `-silent`

source-db-name

Specifies the Progress database you want to copy. You cannot copy the database if you have a server running on it.

target-db-name

Specifies the structure file or the new database. If you specify a directory without a filename, Progress returns an error.

The value you specify can be any combination of letters and numbers, starting with a letter. Do not use Progress 4GL keywords or special characters, such as commas or semicolons. The maximum length of *target-db-name* varies depending on the underlying operating system. For specific limits, see Chapter 3, "Progress Database Limits."

-silent

Suppresses the output of work-in-progress messages.

NOTES

- If you do not supply the .db extension for databases, Progress automatically appends it.
- A target database must contain the same physical structure as the source. For example, it must have the same number of storage areas, records, blocks, and blocksize.
- PROCOPY supports the use of internationalization startup parameters such as `-cpinternal` *codepage* and `-cpstream` *codepage*. See Chapter 18, "Database Startup Parameters," for a description of each database-related internationalization startup parameter.

PRODB Utility

Creates a new Progress database.

PRODB creates a new database from a specified source database. PRODB creates a new database using the structure of the source database and places all of the extents in the current working directory. You can use PRODB to make a copy of any of the demonstration or empty Progress databases.

SYNTAX

Operating System	Syntax

UNIX
Windows
	`prodb [ new-db-name ]`
	`[ empty`
	`| sports`
	`| isports`
	`| sports2000`
	`| old-db-name`
	`| demo`
	`]`

new-db-name

Specifies the name of the database you are creating. If you specify a directory without a filename, Progress returns an error.

The value you specify can be any combination of letters and numbers, starting with a letter. Do not use Progress 4GL keywords or special characters, such as commas or semicolons. The maximum length of *new-db-name* varies, depending on the underlying operating system. For specific limits, see Chapter 3, “Progress Database Limits.”

`empty`

Specifies that the new database is a copy of the `empty` database located in the Progress install directory. PRODB knows where to locate the `empty` database, so you do not need to provide a pathname to it.

In addition to the default `empty` database, PRODB allows you to create other empty database structures with different block sizes:

- empty (default)
- empty1 (1K block size)
- empty2 (2K block size)
- empty4 (4K block size)
- empty8 (8K block size)

To create these `empty` database structures, however, you must specify the pathname to where Progress is installed, or use the DLC environment variable. For example:

UNIX

```
prodb new-db-name $DLC/empty2
```

Windows

```
prodb new-db-name %DLC%\empty2
```

`sports`

Specifies that the new database is a copy of the Progress Sports database.

`isports`

Specifies that the new database is a copy of the Progress international Sports database.

`sports2000`

Specifies that the new database is a copy of the Progress Sports2000 database.

`old-db-name`

Specifies the name of the database you are copying.

`demo`

Specifies that the new database is a copy of the Progress demo database.

NOTES

- You can also create a new database from the Progress Data Dictionary.
- When you use the PRODB utility and give the copy a new name, you cannot run the original r-code against the new database. This is because Progress saves the database with the r-code. To run the r-code against the new database, use the Logical Database Name (`-ld`) startup parameter and use the original database name.
- A new database must contain the same physical structure as the source. For example, it must have the same number of storage areas, records, blocks and blocksize.
- PRODB supports the use of internationalization startup parameters such as `-cpinternal` *`codepage`* and `-cpstream` *`codepage`*. See Chapter 18, "Database Startup Parameters," for a description of each database-related internationalization startup parameter.
- When issuing the PRODB command, specify the target and source database names (they are positional) before specifying an internationalization startup parameter such as `-cpinternal`.
- PRODB also supports the parameter file (`-pf`) startup parameter that contains a list of valid startup parameters.

PRODEL Utility

Deletes a Progress database.

SYNTAX

Operating System	Syntax
UNIX Windows	`prodel` *db-name*

db-name

Specifies the database you are deleting.

NOTES

- When you delete a database, Progress displays a message to notify you that it is deleting the database, log, BI, and AI areas for that database.
- When you delete a database, Progress deletes all associated areas (database, log, and before-image (BI) and after-image (AI) files) that were created using the structure description file.
- If an AI area exists in a database, a warning appears. Back up any AI areas you need before deleting them.
- PRODEL supports the use of internationalization startup parameters such as `-cpinternal` *codepage* and `-cpstream` *codepage*. See Chapter 18, "Database Startup Parameters," for a description of each database-related internationalization startup parameter.

Progress Explorer Utility

Part of a system administration framework that provides a consistent interface for managing all Progress products installed on your network. Within this context, Progress Explorer provides configuration tools by means of a graphical user interface (GUI) for the administration of your Progress database servers. For example, you can use Progress Explorer to perform the following tasks on a local or remote host:

- Connect to an AdminServer
- Start and stop Progress databases
- Configure database properties through a *Property Editor* window.

For instructions on using Progress Explorer to complete these tasks, see the Progress Explorer online help.

To launch Progress Explorer from the Start menu, choose Programs→ Progress→ Progress Explorer Tool.

NOTES

- An AdminServer is installed on every system where you install a Progress database. The AdminServer grants access to each instance of an installed Progress product. The AdminServer must be running in order to use the Progress Explorer configuration tools, or command-line configuration utilities to manage your Progress database. On Windows NT-based systems, the AdminServer starts automatically and runs as an NT service. For UNIX-based systems, a command-line utility (PROADSV) is used to start and stop the AdminServer. For more information about the AdminServer, see the *Progress Installation and Configuration Guide Version 9 for UNIX* or the *Progress Installation and Configuration Guide Version 9 for Windows*.
- The database configurations you create with Progress Explorer are saved in the `conmgr.properties` file. Do not edit the `conmgr.properties` file directly.
- Progress Explorer is a snap-in to the Microsoft Management Console (MMC), the system and network administration tools framework. For more information about MMC, see the MMC online help.

PROLOG Utility

Truncates the log file.

SYNTAX

Operating System	Syntax
UNIX Windows	`prolog` *db-name*

db-name

Specifies the database log you want to truncate. Do not include the .db suffix.

utility-option

Specify print to display the contents of the log file. Specify trim to shorten the log file and save only the last 3,000 bytes.

service-name

Specifies the name of the server. You must specify this parameter only if you specified a server name when you started the server.

printer/filename

Specifies the name of the printer or file to which you want to redirect output when you specify print as the utility option.

NOTES

- If you want to save the log file, use operating system utilities to back it up before using PROLOG.
- See Chapter 5, "Starting Up and Shutting Down," for more information about log files.
- PROLOG supports the use of internationalization startup parameters such as `-cpinternal` *codepage* and `-cpstream` *codepage*. See Chapter 18, "Database Startup Parameters," for a description of each database-related internationalization startup parameter.

PROMON Utility

Starts the Progress database monitor that displays database information.

SYNTAX

Operating System	Syntax
UNIX Windows	promon *db-name*

db-name

Specifies the database you want to monitor.

When you enter the PROMON utility, the main menu appears, as shown in Figure 19–3.

```
                    PROGRESS MONITOR Version 9

                    Database: /usr/wrk/sports2000

                    1. User Control
                    2. Locking and Waiting Statistics
                    3. Block Access
                    4. Record Locking Table
                    5. Activity
                    6. Shared Resources
                    7. Database Status
                    8. Shut Down Database

                    T. Transactions Control
                    L. Resolve Limbo Transactions
                    C. Coordinator Information

                    M. Modify Defaults
                    Q. Quit

                    Enter your selection:
```

Figure 19–3: PROMON Utility Main Menu

The following PROMON entries describe each of these options.

If you choose options 1–4 or option T, the following menu appears:

```
                    1. Display all entries
                    2. Match a user number
                    3. Match a range of user numbers

                    Q. Return to main menu

                    Enter your selection:
```

After the first screen of output from any option, press U to monitor the same parameters continuously. The default interval is eight seconds, but you can change it on the Modify Defaults screen (option M on the main menu).

NOTES

- To collect output data, the monitor bars all users from shared memory for an instant and takes a snapshot of its contents. Therefore, **do not** use the monitor when system performance is being measured.
- Certain monitor output data is not generally useful, but it might be helpful to Progress Software representatives working to debug unusual system problems.

PROMON User Control Option

Displays information for each database process. Figure 19–4 shows an example of this option's output.

```
User Control
Usr   Name     Type      Wait    Trans    PID    Sem   Srv   Login     Time
0               BROK        0        0    3989   3     0     12/20/92  15:38
1     sue       SELF        0     3145    6356   4     0     12/21/92  16:00
2     dave      SELF DB     0        0    6943   5     0     12/22/92  14:45
3               SERV        0        0    6968   6     0     12/22/92  15:00
4     mary      REMC REC   87     3267    7007   7     3     12/22/92  15:28
5     bill      REMC        0        0    7034   9     3     12/22/92  16:01
6               MON         0        0    7123   9     0     12/22/92  16:36
The DB holder is user -1. The MT holder is user -1.

RETUN - repeat, U - continue uninterrupted, Q - quit
```

Figure 19–4: Sample Output For PROMON User Control Option

DISPLAY FIELDS

`Usr`

User numbers are assigned sequentially to database processes as they are started. The broker process is always user (0).

`Name`

The user name for end-user (client) processes.

`Type`

Table 19–2 lists the distinct types of database processes that can appear in the Type field.

Table 19–2: Type Column Values *(1 of 2)*

Value	Process Type
BROK	Broker process
MON	Monitor process
REMC	Remote client

Table 19–2: Type Column Values *(2 of 2)*

Value	Process Type
SELF	Self-service client
SERV	Remote-user server
AIW	After-image writer (AIW) process
SHUT	Database shutdown process

`Wait`

If a process is waiting, there are 10 possible values for the Wait column. Table 19–3 lists the values and resource or event the process is waiting for. (Wait is actually two columns – one textual, the other numeric.)

Table 19–3: Wait Column Values *(1 of 2)*

Value	Name	Resource/Event
REC	Record lock	The process is waiting for a locked record to be freed. The numeric column shows the record ID number of the record the process is waiting for.[1]
SCHM	Schema lock	The process is waiting for the schema lock. There is one schema lock; only one user at a time can update any part of the schema.[1]
TSK	Another task (transaction to end)	The process is waiting to access (or skip) a record that has been marked for deletion, or whose key field value has been changed, by a task (transaction) that has not yet terminated. When the task terminates, this process can resume executing.[1]
DEAD	Process termination	The process has actually terminated, although it still appears in the process list. This can happen if a process is killed with the UNIX `kill -9` command.
BUFF	Database buffer	The process is waiting for a database buffer.[2]

Table 19–3: Wait Column Values *(2 of 2)*

Value	Name	Resource/Event
DB	Database service	The process is waiting for the database server.[2]
IX	Index lock	The process is waiting for an index lock.[2]
MT	Microtransaction	The process is waiting for a microtransaction.[2]
READ	Block read	The process is waiting to read a block.[2]
WRT	Block write	The process is waiting to write a block.[2]

[1] These are application waits. There is no time limit on how long one of these locks can be held.

[2] These are for short-duration (micro- to milliseconds) locks used internally by Progress.

`Trans`

Transaction (task) number, if one is active. After the broker starts, numbers are assigned sequentially to each transaction that accesses the database.

`PID`

The process ID as assigned by the operating system. The PID column typically displays (0) for remote clients.

`Sem`

The number of the semaphore the process is using. Each process uses exactly one semaphore. Progress uses two semaphores (numbers 1 and 2) internally and assigns the remaining semaphores one at a time (starting with the broker) as database processes log in.

`Srv`

For remote clients (REMC), the user number of the server the client runs against.

`Login`

The date the process started.

`Time`

The time the process started.

PROMON Locking and Waiting Statistics Option

Displays lock and wait information for each database process. The first two lines display cumulative statistics (since the broker started) for all users. Figure 19–5 shows an example of this option's output.

```
Locking and Waiting:
Type Usr Name       Record    Trans    Schema
Lock 999 TOTAL...     2345      567       145
Wait 999 TOTAL...       89       12         9
Lock   0               654       32        33
Wait   0                 0        0         0
Lock   1 sue            54        6         3
Wait   1 sue             0        0         0
Lock   2 dave           43        7         2
Wait   2 dave           21        3         2
Lock   3              1123      302        54
Wait   3                12        1         0
Lock   4 mary          345       89        30
Wait   4 mary           20        6         0
Lock   5 bill          101       20        15
Wait   5 bill           10        2         0
Lock   6               175       34         0
Wait   6                15        3         0

RETURN - repeat, U - continue uninterrupted, Q - quit:
```

Figure 19–5: Sample Output For PROMON Locking and Waiting Statistics Option

DISPLAY FIELDS

`Type`

Lock or Wait. The output display includes two lines for each process. The first shows the number of record, transaction, and schema locks obtained, and the second shows the number of times the process waited to obtain one of these locks.

`Usr`

The user number of the process. User number 999 represents all users.

`Name`

For client processes, the user name.

Record

For the Lock type, the total number of record locks obtained. For the Wait type, the number of times the user had to wait to obtain a record lock. Numbers are cumulative for the life of the process.

Trans

For the Lock type, the number of times a transaction lock was issued. A transaction lock is issued for the duration of a transaction when a record is deleted. If the transaction is ultimately undone or backed out, the record is not deleted.

For the Wait type, the number of times the user had to wait before accessing or skipping a record marked for deletion by an unfinished transaction. This transaction wait corresponds exactly to the TSK wait described under User Control (option 1).

Schema

For the Lock type, the total number of times the schema lock was obtained. For the Wait type, the number of times the user had to wait to obtain the schema lock. (There is one schema lock; only one user at a time can update any part of the schema.)

PROMON Block Access Option

Displays database buffer pool access statistics. Figure 19–6 shows an example of this option's output.

```
        Block Access:
                                DB      DB    DB     BI    BI     AI    AI
        Type Usr Name       Reqst   Read  Write  Read  Write  Read  Write
        Acc  999 TOTAL...   17393   1866    424   321    989   204    444
        Acc    0              102    102     34    23     75    12     48
        Acc    1 sue         6060    876    144   165    304    75    128
        Acc    2 dave        4023    654    185    78    406    34    156
        Acc    3               86     84      8    12     22     9     18
        Acc    4 mary        3454      6     13     7     46     3      6
        Acc    5 bill        3726    235     58    43    128    28     71
        Acc    6                0      0      0     0      0     0      0

RETURN - repeat, U - continue uninterrupted, Q - quit:
```

Figure 19–6: Sample Output For PROMON Block Access Option

The first line displays cumulative information for all users. The six read and write columns refer to disk I/O. Reads and writes are always one block. Block size varies among systems, but is usually 512 bytes, 1,024 bytes, or 2,048 bytes.

DISPLAY FIELDS

`Type`

This column always has the value Acc.

`Usr`

The user number of the process.

`Name`

For client processes, the user name.

DB Reqst

The number of times the database buffer system was searched to find a block. The buffer system is searched every time a process attempts to access a record. If the block that holds the desired record is already in memory, a disk read is not required. If the ratio of DB Reqst to DB Read is not high (10 to 1 or greater), consider raising the value of the Blocks in Database Buffers (-B) startup parameter. Regardless of the number of available buffers, random record access causes a lower database request to disk read ratio than sequential record access.

DB Read

The number of database disk block reads. A database block must be read from disk when a process accesses a record whose containing block is not already in the database buffers. Recall that for read-only database requests, Progress uses private database buffers if they are available, rather than the shared buffer pool (allocated with the Blocks in Database Buffers (-B) parameter).

DB Write

The number of database block writes to disk. Once the database buffers are full, every disk read overwrites an existing block; if the overwritten block has been modified since it was read in, it must be written to disk. This accounts for the majority of block writes.

BI Read

The number of before-image (BI) file block reads. For example, the BI file is read when a transaction is undone. The BI file has its own one-block input buffer and does not use the database buffer pool.

BI Write

The number of BI file block writes. When a record is updated, a pretransaction copy of the record is written to the BI file. When the transaction completes, the database engine writes the last BI file block out to disk (assuming you are running the database with full integrity). This post-transaction disk write accounts for the relatively high number of BI file writes, but it can be delayed with the Delay BI File Write (-Mf) startup parameter.

In addition to record images, the database engine writes to the BI file various notes and data required to reconstruct a damaged database. The BI file has its own one-block output buffer and does not use the shared database buffer pool.

`AI Read`

The number of after-image (AI) file block reads. The AI file is read during crash recovery. The AI file has a one-block input/output buffer and does not use the database buffer pool.

`AI Write`

The number of AI file block writes. When you run the database with after-imaging enabled, a copy of each note written to the BI file is written to the AI file.

PROMON Record Locking Table Option

Displays the contents of the record locking table. Each line corresponds to an entry in the locking table. The lock table contains entries for all record locks currently granted or requested by users of the database. Figure 19–7 shows an example of this option's output.

```
Record Locking Table:
Usr Name      Chain #       Rec-id Lock Flags
  1 sue       REC   1         5439 SHR
  1 sue       REC   3         5874 EXCL     Q
  2 dave      REC   0         5568 EXCL
  2 dave      REC   2         6002 SHR      L
  2 dave      REC   3         5939 SHR
  2 dave      REC   8         5736 SHR      P
  3           REC   2         4873 EXCL
  3           REC   3         5238 SHR      U
  4 mary      REC   0         5293 SHR
  4 mary      REC   3         4247 SHR
  4 mary      REC   6         3984 SHR
  5 bill      REC   1         6324 EXCL
  5 bill      REC   2         3746 SHR      P
  5 bill      REC   4         5233 NOLK
  5 bill      REC   6         5683 EXCL
  5 bill      REC   9         4832 SHR
  5 bill      REC  11         5293 SHR

RETURN - repeat, U - continue uninterrupted, Q - quit
```

Figure 19–7: Sample Output For PROMON Record Locking Table Option

The size of the record locking table is set with the Locking Table Entries (-L) startup parameter. See the chapter on locks in the *Progress Programming Handbook* for more information on locks and locking.

DISPLAY FIELDS

`Usr`

The user number.

`Name`

For client processes, the user name.

`Chain`

The chain type should always be REC, the record lock chain.

`#`

The record lock chain number. The locking table is divided into chains anchored in a hash table. These chains provide for fast lookup of record locks by Rec-id.

`Rec-id`

The record ID for the lock table entry. The Rec-id column identifies the records locked by each database process.

`Table`

The ID of the locked table.

`Lock`

One of five lock types: X (exclusive lock), S (share lock), IX (intent exclusive lock), IS (intent sharelock), or SIX (shared lock on table with intent to set exclusive locks on records).

`Flags`

There are five possible types of flags: L, P, Q, or U. Table 19–4 lists the flags and their meanings.

Table 19–4: Flag Values

Flag	Name	Description
L	Limbo lock	The client has released the record, but the transaction has not completed. The record lock is not released until the transaction ends.
P	Purged lock entry	The lock is no longer held.
Q	Queued lock request	Represents a queued request for a lock already held by another process.
U	Upgrade request	The user has requested a lock upgrade from SHARE to EXCLUSIVE but is waiting for another process that holds a conflicting lock.
H	No hold	The “nohold” flag is set.

PROMON Activity Option

Displays an overview of your system activity. Figure 19–8 shows an example of this option's output.

```
Activity - Sampled at 03/10/02 11:25 for 2:46:37.

Event           Total   Per Sec       Event            Total         Per Sec
Commits          3900       0.3        Undos             1215             0.1
Record Updates   4717       0.4        Record Reads    353418            35.3
Record Creates  12307       1.2        Record Deletes   10837             1.0
DB Writes       29778       2.9        DB Reads        233257            23.3
BI Writes       27834       2.7        BI Reads         10592             1.0
AI Writes       29443       2.9
Record Locks   113928      11.0        Record Waits         0             0.0
Checkpoints      1591       0.1        Buffers Flushed  21154             2.1

Rec Lock Waits   0 %     BI Buf Waits     2 %     AI Buf Waits       0 %
Writes by APW   23 %     Writes by BIW   85 %     Writes by AIW     88 %
Buffer Hits     87 %
DB Size       1520 K        BI Size      64 K         AI Size        0 K
FR chain        16 blocks   RM chain    733 blocks
Shared Memory  184 K        Segments      1

0 Servers, 1 Users (1 Local, 0 Remote, 0 Batch), 0 Apws

RETURN - repeat, U - continue uninterrupted, Q - quit:
```

Figure 19–8: Sample Output For PROMON Activity Option

DISPLAY FIELDS

Event

The events that have occurred on the system and the cumulative total number of events and the number of events per second. Table 19–5 defines the event types listed in the Event field. For each event type, PROMON lists the cumulative total number of events and the number of events per second.

Table 19–5: Event Types

Event Type	Description
Commits	Number of transactions committed.
Undos	Number of transactions rolled back.
Record Updates	Number of records updated.
Record Reads	Number of records read.
Record Creates	Number of records created.
Record Deletes	Number of records deleted.
DB Writes	Number of database blocks written to disk.
DB Reads	Number of database blocks read from disk.
BI Writes	Number of BI blocks written to disk.
BI Reads	Number of BI blocks read.
AI Writes	Number of AI blocks written to disk.
Record Locks	Number of record locks used.
Record Waits	Number of times the database engine waited to access a locked record.
Checkpoints	Number of checkpoints performed.
Buffers Flushed	Number of database buffers flushed during checkpoints.

`Rec Lock Waits`

Percentage of record accesses that result in record lock waits. A record lock wait occurs when the database engine must wait to access a locked record.

`BI Buf Waits`

Percentage of before-image (BI) buffer waits. A BI buffer wait occurs when the database engine must wait to access a BI buffer.

`AI Buf Waits`

Percentage of after-image (AI) buffer waits. An AI buffer wait occurs when the database engine must wait to access an AI buffer.

`Writes by APW`

Percentage of database blocks written to disk by the APW; this is a percentage of the total number of database blocks written by the database engine.

`Writes by BIW`

Percentage of BI blocks written to disk by the BIW; this is a percentage of the total number of BI blocks written to disk by the database engine.

`Writes by AIW`

Percentage of AI blocks written to disk by the AIW; this is a percentage of the total number of AI blocks written by Progress.

`Buffer Hits`

Percentage of buffer hits. A buffer hit occurs when Progress locates a database record in the buffer pool and does not have to read the record from disk.

`DB Size`

Size of your database in kilobytes.

`BI Size`

Size of your BI file in kilobytes.

`AI Size`

Size of your AI file in kilobytes.

`FR chain`

Number of blocks on your database's free chain. The free chain is a chain of empty database blocks.

`RM chain`

Number of blocks on your database's record-management chain. The record-management chain is a chain of partially filled database blocks.

`Shared Memory`

Amount of shared memory used by the database, in kilobytes.

`Segments`

Number of shared-memory segments allocated by the database engine. The size of a shared-memory segment is determined by your kernel configuration.

The last line of the PROMON Activity output summarizes the current number of each type of process running against the database at the time you ran the Activity option, not a cumulative total. Table 19–6 defines the process types.

Table 19–6: Process Types

Field	Description
Servers	Number of servers running against your database. This is the current value at the time you ran the Activity option, not a cumulative total.
Users	Number of users running Progress. This field is further divided into the type of user: Local, Remote, or Batch. This is the current value at the time you ran the Activity option, not a cumulative total.
Apws	Number of APWs running against your database. This is the current value at the time you ran the Activity option, not a cumulative total.

PROMON Shared Resources Option

Displays a variety of system resource usage statistics and startup parameter settings. Broker status information appears only when a broker process is active. Figure 19–9 shows this option's output. See Chapter 18, "Database Startup Parameters," for more information on these startup parameters.

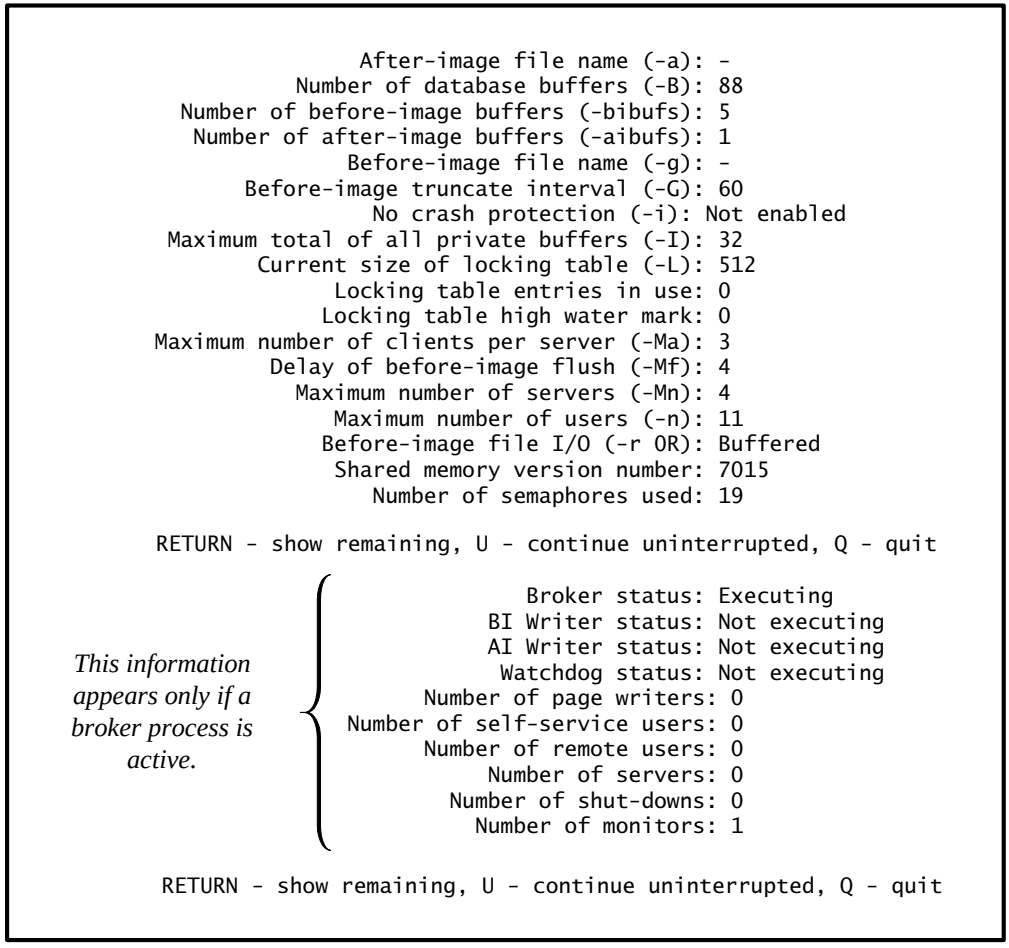


Figure 19–9: Sample Output For PROMON Shared Resources Option

DISPLAY FIELDS

Because most of the Shared Resources options output is self-explanatory, this list describes only two items:

`Shared memory version number`

The version number of the shared-memory data structure. This structure varies slightly between some versions of Progress. Since broker, server, and client processes all access the same shared-memory pool, their shared-memory version numbers must agree. If an error message states that shared-memory version numbers do not match, make sure that the broker and clients are running the same version of Progress.

`Number of semaphores used (UNIX only)`

On UNIX systems, shows the number of semaphores preallocated for this database. Because each process requires one semaphore, semaphores are preallocated based on the number of processes expected to access the database. For a particular database, the number of semaphores used (#SEM) is as follows:

#SEM = *Max-possible-users* (-n) + *Max-possible-servers* (-Mn) + 4

The maximum number of semaphores you can use on your system is set with a kernel configuration parameter (SEMMNS). If the number of database users grows large, you can exceed this value. You must reconfigure the system's kernel to increase the semaphore limit. (See *Chapter 14, "Managing Performance."*) You can only reduce semaphore use by lowering the values of the Number of Users (`-n`) or Maximum Servers (`-Mn`) startup parameters. (When you start the broker process, a message specifies the number of semaphores still available.)

PROMON Database Status Option

Displays database block usage statistics and general status information. Figure 19–10 shows an example of this option's output.

```
                    Database version number: 1085
                             Database state: Open (1)
                      Database damaged flags: None (0)
                             Integrity flags: Executing with -r (4)
                 Database block size (bytes): 1024
               Total number of database blocks: 208
               Database blocks high water mark: 208
             Free blocks below high water mark: 0
                 Record blocks with free space: 0
               Before image block size (bytes): 1024
                Before-image cluster size (kb): 128
                After image block size (bytes): 1024

                          Last transaction number: 17923
                      Highest file number defined: 10
                           Database Character Set: ISO 8859-1
                          Database Collation Name: Basic
                Database create (multi-volume): - -
                     Most recent database open: 6/12/01 15:38
                        Previous database open: 6/10/01 8:31

RETURN - show remaining, U - continue uninterrupted, Q - quit:

                  Local Schema cache time stamp: 02/15/02 21:41
                      Most recent .bi file open: 02/17/02 17:01
                         Previous .bi file open: - -
                    Time since last truncate bi: 0 (seconds)
                        Most recent full backup: 02/15/02 21:43
                  Database changed since backup: No
                 Most recent incremental backup: - -
                   Sequence of last incremental: 0

RETURN - repeat, U - continue uninterrupted, Q - quit
```

Figure 19–10: Sample Output For PROMON Database Status Option

DISPLAY FIELDS

Because most of the Database Status option output is self-explanatory, this list describes only five items:

`Database state`

One of the following values, set in the database master block:

Database State	Description
1	The database is open.
2	The database is closed, used on disk.
3	The database is being recovered.

`Total number of database blocks`

The actual database size in disk blocks.

`Number of empty blocks`

The number of empty blocks in the database. Empty blocks are created when the database outgrows its existing block allocation, and also by record and index deletion. Progress expands the database by multiple blocks at a time—usually 8 or 16—and most of these are empty at first. Typically, the number of empty blocks increases and decreases as the database grows.

`Record blocks with free space`

Free space is created when records are deleted and when the database grows into a new block, but does not use the entire block. Progress reuses empty space with new records when possible, but does not return empty space to the operating system. This space does not affect Progress performance. However, if you must reclaim this disk space for your system, you can dump and reload the database.

`Time since last truncate bi`

The time in seconds since the before-image (BI) file was truncated. When the BI file is truncated, all modified database blocks are written to disk.

PROMON Shut Down Database Option

Lets you disconnect a user or shut down a database. When you choose this option, PROMON displays the menu shown in Figure 19–11.

```
                Enter your selection:

                1 Disconnect a User
                2 Unconditional Shutdown
                3 Emergency Shutdown (Kill All)
                x Exit

                Enter choice>
```

Figure 19–11: Sample Output For PROMON Shut Down Database Option

MENU CHOICES

`Disconnect a User`

Disconnects a single user from the database specified when PROMON was invoked. PROMON prompts you to enter the user number of the user you want to disconnect. (PROMON displays the user number of active users above this submenu.) Enter the user number, then press **RETURN**.

`Unconditional Shutdown`

Shuts down the database in an orderly fashion. When you choose Unconditional Shutdown, PROMON begins the shutdown when you press **RETURN**; PROMON does not prompt you to confirm the shutdown.

`Emergency Shutdown (Kill All)`

Shuts down the database without performing cleanup and kills all processes connected to the database. Before performing the shutdown, PROMON displays the following message prompting you to confirm the shutdown:

```
This is rather extreme. All the processes connected to
the database will be destroyed. Do you really want to?
Please confirm by entering y for yes:
```

If you enter anything other than **y** or **yes**, PROMON cancels the shutdown and returns to the PROMON utility main menu.

PROMON Transactions Control Option

Displays information about distributed transactions in a database. Figure 19–12 shows an example of this option's output.

```
Transaction Control:
Usr  Name   Trans   Login   Time   R-comm?   Limbo?   Crd?   Coord  Crd-task
  2  paul   757  07/25/01   10:15    yes       yes      no    sports1   760
       .     .      .       .          .    .          .         .      .
       .     .      .       .          .    .          .         .      .
       .     .      .       .          .    .          .         .      .


RETURN - repeat, U - continue uninterrupted, Q - quit
```

Figure 19–12: Sample Output For PROMON Transaction Control Option

NOTE: If you run PROMON against a database where no limbo transaction has occurred, Progress does not display any field information on the Transaction Control screen.

DISPLAY FIELDS

`Usr`

The number of the user running the distributed transaction.

`Name`

The name of the user running the distributed transaction.

`Trans`

The transaction number on the machine where you are running PROMON.

`Login`

The date the user running the distributed transaction began the Progress session.

`Time`

The time the user running the distributed transaction began the Progress session.

`R-comm?`

The state of the transaction. If this field is yes, the transaction is in a ready-to-commit state. This means that the first phase of two-phase commit is complete.

`Limbo?`

Whether a transaction is a limbo transaction. If one or more limbo transactions occur, the Limbo field displays a yes for each transaction listed. A yes in this field means that you must resolve transactions.

`Crd?`

Whether the database you are running PROMON against is the coordinator.

`Coord`

The name of the coordinator database.

`Crd-task`

The transaction number of the coordinator database.

PROMON Resolve Limbo Transactions Option

Lets you resolve a limbo transaction by either committing or aborting the transaction. When you choose this option, PROMON displays the menu shown in Figure 19–13.

```
1 Abort a Limbo Transaction
2 Commit a Limbo Transaction
Q Quit

Enter choice>
```

Figure 19–13: Sample Output For PROMON Resolve Limbo Transactions Option

MENU CHOICES

`Abort a Limbo Transaction`

When you choose Abort a Limbo Transaction, Progress prompts you to enter the user number of the transaction you want to abort. (PROMON displays the user's number in the User column of the Transaction Control option screen.) Enter the user's number, then press **RETURN**.

`Commit a Limbo Transaction`

When you choose Commit a Limbo Transaction, PROMON prompts you to enter the user number of the transaction you want to commit. (PROMON displays the user number in the Usr column of the Transaction Control option screen.) Enter the user number, then press **RETURN**. PROMON displays a message similar to the following:

```
User 1: commit transaction and disconnect.
```

NOTE: To commit transactions on a database that is shut down, you must use the 2PHASE RECOVER qualifier of PROUTIL.

PROMON Coordinator Information Option

Lets you determine whether a limbo transaction was committed. You need this information before resolving a limbo transaction.

To get this information, run PROMON against the transaction's coordinator database. PROMON displays the name of a transaction's coordinator database in the Coord column of the Transaction Control option screen.

NOTE: If the coordinator database is shut down and you cannot run PROMON against it, you must use the 2PHASE COMMIT option of PROUTIL to determine whether it committed the transaction.

Figure 19–14 shows an example of this option's output.

```
PROGRESS MONITOR Version 9.0

    Database: /users/sports1

    Q. QUIT

Enter the transaction number you want to find out if committed:
```

Figure 19–14: Sample Output For PROMON Coordinator Information Option

PROMON displays the transaction number in the Crd-task column of the Transaction Control option screen. Enter the transaction number and press **RETURN**. Progress displays a message that tells you whether the transaction committed.

PROMON Modify Defaults Option

Displays terminal behavior parameters and lets you change them when running the monitor. It also displays and lets you change the value of startup parameters used for tuning the asynchronous page writer.

Figure 19–15 shows the menu that appears when you choose the Modify Defaults option.

```
     Enter your selection: m

     1. Page size:      24
     2. Clear screen for first page: Yes
     3. Short pause after each page: 4
     4. Long pause after last page: 8
     5. Monitor Sampling Interval:  30 sec.
     6. APW queue delay:    500 ms.
     7. APW queue start:    1
     8. APW scan delay:   3 sec.
     9. APW scan count:   9
     a. APW write limit:    5

     Q. Return to main menu

     Enter your selection:
```

Figure 19–15: PROMON Modify Defaults Menu

DISPLAY FIELDS

`Page size`

The number of lines on the terminal screen.

`Clear screen for first page`

If Yes, the system clears the screen before the main menu and before every list.

`Short pause after each page`

The delay in seconds between pages of output when in continuous monitoring mode. (Enter **u** for the Continue Uninterrupted option.)

`Long pause after last page`

The delay after the last page of output when in continuous monitoring mode.

For example, if you choose option 7 (Database Status), wait for first page of output, and enter **u**, then by default:

- There is a four-second delay before each remaining page of output appears.
- There is an eight-second delay before database statistics reappear.

`Monitor Sampling Interval`

The interval at which PROMON samples data, in seconds.

`APW queue delay`

The value of the Page Writer Queue Delay (`-pwqdelay`) startup parameter. `-pwqdelay` is a self-tuning parameter. When the database encounters heavy loads, `-pwqdelay` decreases its value so that the APW writes more often. When the demand on the database lessens, `-pwqdelay` increases its value so that the APW writes less often. The default value, in milliseconds, for `-pwqdelay` is 100.

`APW queue start`

The value of the Page Writer Queue Minimum (`-pwqmin`) startup parameter. Do not change this value unless directed to do so by Progress Software Technical Support.

`APW scan delay`

The value of the Page Writer Scan Delay (`-pwsdelay`) startup parameter. The default value, in seconds, of `-pwsdelay` is 1.

`APW scan count`

The value of the Page Writer Scan (`-pwscan`) parameter. Do not change this value unless directed to do so by Progress Software Technical Support.

`APW write limit`

The value of the Page Writer Maximum Buffers (`-pwwmax`) parameter. Do not change this value unless directed to do so by Progress Software Technical Support.

PROREST Utility

Verifies the integrity of a database backup or restores a full or incremental backup of a database.

SYNTAX

Operating System	Syntax
UNIX Windows	`prorest` *db-name device-name* **{** `-vp` **|** `-vf` **|** `-list` **}**

db-name

Specifies the database where you want to restore the backups.

device-name

Identifies the directory pathname of the input device (for example, a tape drive) or a standard file from which you are restoring the data. If *device-name* identifies a block or character special device, Progress assumes the device has removable media, such as a tape or a floppy diskette.

`-vp`

Specifies that the restore utility reads the backup volumes and computes and compares the backup block cyclic redundancy checks (CRCs) with those in the block headers. To recover any data from a bad block, you must have specified a redundancy factor when you performed the database backup. See Chapter 7, "Backing Up a Database," for more information about error correction blocks and data recovery.

`-vf`

Specifies that the restore utility only compares the backup to the database block for block.

`-list`

Provides a description of all application data storage areas contained within a database backup. Use the information to create a new structure description file and database so you can restore the backup.

NOTES

- The PROREST utility has been enhanced to use only Version 9 databases. There is no ability to read or write backup tapes from previous versions of the Progress database.

When restoring a backup, the target database must contain the same physical structure as the backup version. For example, it must have the same number of storage areas, records per block, and blocksize.

- PROREST restores transaction log (TL) extents.
- Before you restore a database, you might want to verify that your backup does not contain any corrupted blocks. You can use the PROREST utility to verify the integrity of a full or incremental backup of a database by using the Partial Verify or Full Verify parameters.
- The Partial Verify or Full Verify parameters do not restore or alter the database. You must use the PROREST utility separately to restore the database.
- You can use the Partial Verify parameter with both online and offline backups.
- Use the Full Verify parameter immediately after performing an offline backup to verify that the backup is correct.
- PROREST supports the use of internationalization startup parameters such as `-cpinternal` *codepage* and `-cpstream` *codepage*. See Chapter 18, "Database Startup Parameters," for a description of each database-related internationalization startup parameter.

PROSTRCT Utility

Creates and maintains a Progress database. For example, you can use PROSTRCT and its qualifiers to perform the following tasks:

- Create a new database from an existing structure description (ST) file.
- Display storage usage statistics including information about storage areas in a database.
- Add areas and extents to a database.
- Remove areas and extents from a database.

SYNTAX

Operating System	Syntax
UNIX Windows	`prostrct` *qualifier db-name* **[** *structure-description-file* **]**

qualifier

Specifies the qualifier that you want to use. You can supply the following qualifiers:

- ADD
- BUILDDB
- CREATE
- LIST
- REMOVE
- REPAIR
- STATISTICS
- UNLOCK

Details of the above qualifiers are found in the PROSTRCT subsections listed in the following pages.

db-name

Specifies the database you are using.

structure-description-file

Specifies the new structure description (ST) file. Do not use `dbname.st`.

NOTES

- You must define your database structure in a structure description (ST) file before you create a database. PROSTRCT CREATE creates a database control (DB) area from the information in the ST file.
- See Chapter 4, "Creating and Deleting Databases," for a complete description of structure description (ST) files and storage areas.
- PROSTRCT supports the use of internationalization startup parameters such as `-cpinternal` *codepage* and `-cpstream` *codepage*. See Chapter 18, "Database Startup Parameters," for a description of each database-related internationalization startup parameter.

PROSTRCT ADD Qualifier

Appends the files from a new structure description (ST) file to a Progress database. For example, you can use PROSTRCT ADD to add new storage areas and extents to new or existing storage areas.

SYNTAX

Operating System	Syntax
UNIX Windows	`prostrct add` *db-name* **[** *structure-description-file* **]**

db-name

Specifies the database you are using.

structure-description-file

Specifies the new structure description (ST) file. Do not use *db-name*`.st`.

NOTES

- You can use PROSTRCT ADD to add areas and extents only when the database is offline.
- The new structure description file cannot identify existing extent files. It can only contain the definitions for new extent files. See Chapter 4, "Creating and Deleting Databases," for a complete description of structure description (ST) files and storage areas.

PROSTRCT BUILDDB Qualifier

Re-creates a control area from the structure description (ST) of an existing database. Use to recover when an existing database control (DB) area is lost or damaged.

SYNTAX

Operating System	Syntax
UNIX Windows	`prostrct builddb` *db-name* [*structure-description-file*]

db-name

Specifies the database you are using.

structure-description-file

Specifies the existing structure description (ST) file.

NOTES

- BUILDDB does only minimal validation of the resulting control area.

PROSTRCT CREATE Qualifier

Creates a void Progress database from a previously defined structure description (ST) file. The newly created database does not contain any Progress metaschema information. Rather, it consists of the database control (DB) area and whatever primary recovery (BI), after-image (AI), two-phase commit transaction log (TL), and application data (D*n*) areas you defined in the ST file.

After you create a void database, you must add metaschema information. Therefore, Progress provides empty databases each the size of a supported database block size. The empty database and the database you want to copy it to must have the same block size.

NOTE: Never use operating system file commands to copy a Progress Version 9 database. Instead, use the PROCOPY or PRODB utilities.

SYNTAX

Operating System	Syntax
UNIX Windows	`prostrct create` *db-name* [*structure-description-file*] [`-blocksize` *blocksize*]

db-name

Specifies the Progress database you want to create.

structure-description-file

Specifies the structure description (ST) file you want Progress to access for file information.

-blocksize

Specifies the Progress database block size in bytes (for example `-blocksize 1024`). The maximum number of indexes allowed in a database is based on the database blocksize. For more information of database limits, see Chapter 3, “Progress Database Limits.”

NOTES

- The default structure description (ST) file is `db-name.st`. If you have a structure description file that has the same name as the database you are creating, you do not have to specify the *`structure-description-file`*. Progress automatically creates the database from the structure description file that has the same name as your database with the extension .st.
- You cannot create a Progress database if one already exists with the same name.
- If you use the `-blocksize` parameter, you must place it as the last parameter on the command line.
- The PROSTRCT CREATE utility allows you to specify the minimum amount of information necessary to create a database. You must specify the area type and extent location. If the extent is fixed length, you must also specify the size. You need not provide specific filename or file extensions. The utility will generate filename and file extensions for all database files according to the following naming convention:
  - If the pathname is for a raw device, the name is taken as-is, with no changes.
  - If a relative pathname is provided, including using common "." (dot) notation, the relative pathname will be expanded to an absolute pathname.
  - For BI extents, the filename is the database name with a .b*n* extension, where *n* represents the number of extents created.
  - For AI extents, the filename is the database name with a .a*n* extension, where *n* represents the logical order in which the AI areas will be accessed during normal processing.
  - For TL extents, the filename is the database name, with a .tl extension.
  - For Schema area extents, the filename is the database name with a .d*n* extension, where *n* represents the order extents were created and will be used.
  - For application data extents, the filename is the database name and an area number. The area number is a unique identifier set by Progress to differentiate between different areas. User extent filenames also have a .d*n* extension, where *n* represents the order extents were created and will be used.
- See Chapter 9, "Maintaining Database Structure," for more information about the CREATE qualifier with the PROSTRCT utility.

PROSTRCT LIST Qualifier

Creates a structure description (ST) file for a Progress database. It provides storage area, transaction log, and records per block information in the structure description (ST) file it produces. Also, PROSTRCT LIST displays storage area names and extent information including the extent type, size, number, and name.

SYNTAX

Operating System	Syntax
UNIX Windows	`prostrct list` *db-name* [*structure-description-file*]

db-name

Specifies the multi-volume database whose structure description file you want to update.

structure-description-file

Specifies the structure description file Progress creates. If you do not specify the structure description file, Progress uses the base name of the database and appends a .st extension. It replaces an existing file of the same name.

NOTES

- You can use this utility with an online database.
- Use PROSTRCT LIST any time you make changes to the structure of the database to verify that the change was successful.
- See Chapter 9, "Maintaining Database Structure," for more information about the LIST qualifier with the PROSTRCT utility.

PROSTRCT REMOVE Qualifier

Removes storage areas or extents within storage areas.

SYNTAX

Operating System	Syntax
UNIX Windows	`prostrct remove` *db-name extent-token storage-area*

db-name

Specifies the database where you want to remove a storage area or extent.

extent-token

Indicates the type of extent to remove. Specify one of the following:

- d, to remove a data extent
- bi, to remove a BI extent
- ai, to remove an AI extent
- tl, to remove a TL extent

storage-area

Specifies the storage area to remove.

NOTES

- You cannot remove an extent if it is in use or contains any data.
- Before you remove extents from the transaction log (TL) area, you must disable two-phase commit.
- You must disable after-imaging to remove variable, or fixed-length AI extents.
- You can verify that the deletion occurred and update the structure description file using the LIST qualifier with the PROSTRCT utility.
- See Chapter 9, "Maintaining Database Structure," for more information about the REMOTE qualifier with the PROSTRCT utility.

PROSTRCT REPAIR Qualifier

Updates a database's control information after an extent is moved or renamed.

SYNTAX

Operating System	Syntax
UNIX Windows	`prostrct repair` *db-name* [*structure-description-file*]

db-name

Specifies the database you are using.

structure-description-file

Specifies the structure description (ST) file containing the updated extent information. If you omit the *structure-description-file*, PROSTRCT REPAIR uses the *db-name*`.st` file to update the control information.

NOTES

- Start with a current copy of the database ST file.
- The .st extension on the new structure description file is optional.
- You cannot use the REPAIR qualifier to add or remove extents. You can only use it to change the location of existing extents.
- You must manually move the .db file or the data extent. PROSTRCT REPAIR simply updates the file list of the .db file to reflect the new locations of database extents.

PROSTRCT STATISTICS Qualifier

Displays the following information:

- Database name
- Primary database block size and the before-image and after-image block sizes
- Storage area and each extent within it
- Total number of active blocks allocated for each data area
- Total number of blocks for each data area
- Total number of empty blocks in each data area
- Total number of extent blocks for each data area
- Total records per block for each data area
- Total number of all blocks (active blocks, active free blocks, and empty blocks)
- Date and time of the last full database backup

SYNTAX

Operating System	Syntax
UNIX Windows	`prostrct statistics` *db-name*

db-name

Specifies the database whose storage information you want.

NOTES

- Do not use PROSTRCT STATISTICS when a database is in use.
- Do not run this utility against a database that has crashed. You must recover the database first.
- A void database cannot be accessed by this command.
- See Chapter 9, “Maintaining Database Structure,” for more information about the STATISTICS qualifier with the PROSTRCT utility.

PROSTRCT UNLOCK Qualifier

Allows you to access a damaged database structure and correct inconsistencies between the creation date and the open date in the database file header block and the master block. This is especially useful when trying to force open a database or when recovering a lost database file.

CAUTION: Use this utility as a last resort. Call Progress Software Technical Support for further direction.

SYNTAX

Operating System	Syntax
UNIX Windows	`prostrct unlock` *db-name* **[** `-extents` **]**

db-name

Specifies the database where you want to force access.

-extents

Replaces missing extents with empty extents if any database files are missing.

NOTES

- When Progress finds an inconsistency among the data and recovery log, it generates an error message and stops any attempt to open the database. Typically, inconsistencies between files are a result of accidental misuse of operating system copy utilities, deletion mistakes, or incorrectly administered backup and restore procedures.
- If the first data file (.d1) is missing, the database cannot open because of the missing master block. PROSTRCT UNLOCK with the `-extents` parameter, however, creates an empty file with the same name and location as the missing file that allows the database to open. This function helps enable access to a severely damaged database.
- PROSTRCT UNLOCK does not repair damaged databases. It opens databases with inconsistencies in dates and missing extents, but these databases still need to be repaired before they can be used. For information on repairing databases, see the "Restoring a Database" section in Chapter 7, "Backing Up a Database."

PROUTIL Utility

Performs various database operations, depending on the parameters and qualifiers you supply.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *`db-name`* `-C` *`qualifier`*

`db-name`

Specifies the database you are using.

`-C`

Specifies a particular utility or function when you use PROUTIL.

`qualifier`

Specifies the qualifier that you want to use. You can supply the qualifiers described in Table 19–7.

NOTE: PROUTIL and its qualifiers support the use of internationalization startup parameters such as `-cpinternal` *`codepage`* and `-cpstream` *`codepage`*. See Chapter 18, "Database Startup Parameters," for a description of each database-related internationalization startup parameter.

Table 19–7: PROUTIL Utility Qualifiers *(1 of 4)*

Qualifier	Description
2PHASE BEGIN	Enables two-phase commit for a Progress database.
2PHASE COMMIT	Determines whether the coordinator database committed a transaction.
2PHASE END	Disables two-phase commit for a Progress database.
2PHASE MODIFY	Changes the nickname or priority of a Progress database for two-phase commit.
2PHASE RECOVER	Commits or aborts limbo transactions for a database.

Table 19–7: PROUTIL Utility Qualifiers *(2 of 4)*

Qualifier	Description
BIGROW	Specifies the number of BI clusters available to a database.
BULKLOAD	Loads data files into a database.
BUSY	Determines whether the database is currently in use.
CHANALYS	Displays information about free chain blocks.
CODEPAGE-COMPILER	Compiles a conversion map file from text format to binary format.
CONV89	Converts a Version 8 database to a Version 9 database.
CONVCHAR	Converts a database's character set or identifies a database's character set (for undefined databases).
CONVFILE	Converts a text file from one character set to any other character set.
DBANALYS	Displays statistical information about index, record, and free chain blocks.
DBAUTHKEY	Sets an authorization key for a database.
DBIPCS	Displays status information for shared-memory segments attached by all Progress databases on the system.
DUMP	Performs a binary dump, which is generally faster that an ASCII dump.
DUMPSPECIFIED	Performs a selective binary dump and allows you to dump by field value.
ENABLELARGEFILES	Enables large file processing for a database.
ENABLEPDR	Enables database replication with Peer Direct.
HOLDER	Determines whether the database is currently in use in single-user mode, multi-user mode, or by a Progress utility.
IDXANALYS	Displays information on index blocks.

Table 19–7: PROUTIL Utility Qualifiers *(3 of 4)*

Qualifier	Description
IDXCOMPACT	Performs index compaction online and increases space utilization of the index block to a specified compacting percentage.
IDXMOVE	Moves an index from one application data area to another while the database remains online.
IDXBUILD	Consolidates index records to use disk space as efficiently as possible. Activates deactivated indexes in the database. Repairs corrupted indexes in the database.
IDXCHECK	Checks Progress database indexes to determine whether an index is corrupt, and if it is, diagnoses the problem.
IDXFIX	Checks the Progress database records and indexes to determine whether an index is corrupt or a record has a missing or incorrect index. You can scan the database, the indexes, or both.
IOSTATS	Provides current statistics for active databases. The statistics include buffered, unbuffered, and logical I/O database operations. The statistics are cumulative from database startup.
LOAD	Performs a binary load.
MVSCH	Moves schema after a database conversion.
RCODEKEY	Inserts the authorization key into existing CRC-based r-code.
TABANALYS	Displays information about the degree of fragmentation for each table in a database.
TABLEMOVE	Moves a table and optionally its associated indexes from one storage area to another while the database remains online.
TRUNCATE AREA	Truncates application data storage areas in the specified database. Use this qualifier before you remove storage areas and extents.

Table 19–7: PROUTIL Utility Qualifiers *(4 of 4)*

Qualifier	Description
TRUNCATE BI	Performs three functions: Uses the information in the before-image (BI) files to bring the database up to date, waits to verify that the information has been successfully written to the disk, then truncates the before-image file to its original length. Sets the BI cluster size using the Before-image Filename (`-bi`) parameter. Sets the BI block size using the Before-image Block Size (`-biblocksize`) parameter.
UPDATEVST	Loads the specified database with the most recent virtual system tables.
WBREAK- COMPILER	Compiles the word-break table.
WORD-RULES	Compiles a word-rules file that tells Progress that the rules in the specified files apply to a database.

The following PROUTIL entries describe each of these qualifiers. See the appropriate section for more detailed information.

PROUTIL 2PHASE BEGIN Qualifier

Enables two-phase commit for a Progress database.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C 2phase begin` **[** `-crd` **|** `-tp` *nickname* **]** **...**

db-name

Specifies the database you are using.

`-crd`

Specifies that the database receive top priority when Progress assigns a coordinator database. If you do not specify a database, Progress randomly assigns a coordinator database from the available databases.

Specifying a single coordinator database can greatly simplify the process of resolving limbo transactions because Progress does not have to consult several different coordinating databases.

`-tp` *nickname*

Identifies a unique nickname or alternate name that Progress uses to identify the coordinator database. If you do not specify a nickname, Progress automatically assigns the name of the database without the .db extension as the nickname.

Specifying a nickname simplifies resolving limbo transactions when two databases have the same pathname but are on different machines.

NOTES

- You cannot use the PROUTIL 2PHASE BEGIN qualifier against a database that is online.
- A TL (Transaction Log) area is required to hold the transaction log data generated when two-phase commit is in use. You must create and maintain a TL (Transaction Log) area for your database in order to use two-phase commit.

PROUTIL 2PHASE COMMIT Qualifier

Determines whether the coordinator database committed a transaction.

SYNTAX

Operating System	Syntax
UNIX Windows	proutil *db-name* -C 2phase commit *tr-number*

db-name

Specifies the coordinator database.

tr-number

Specifies the number of the transaction where you want to check the commit status.

If the coordinator committed the transaction, Progress displays a message similar to the following:

```
Transaction 768 has committed. (2048)
```

Once you determine that the coordinator database committed the transaction, you must also commit the transaction on the database where the limbo transaction occurred. If the coordinator did not commit the transaction, you must terminate the transaction on the database where the limbo transaction occurred.

PROUTIL 2PHASE END Qualifier

Disables two-phase commit for a Progress database.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C 2phase end`

db-name

Specifies the database where you want to disable two-phase commit.

PROUTIL 2PHASE MODIFY Qualifier

Changes the nickname or priority of a Progress database for two-phase commit.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C 2phase modify [ -crd | -tp` *nickname* `] ...`

db-name

Specifies the database you are using.

`-crd`

Switches whether or not the database can serve as a coordinator database. If you specify `-crd` against a database that is a candidate for coordinator database, it is no longer a candidate. If you specify `-crd` against a database that is not a candidate, it becomes a candidate.

`-tp` *nickname*

Identifies a new nickname for the coordinator database.

Specifying a nickname simplifies resolving limbo transactions when two databases have the same pathname but are on different machines.

PROUTIL 2PHASE RECOVER Qualifier

Commits or aborts limbo transactions for a database.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C 2phase recover`

db-name

Specifies the database with the limbo transaction.

When you run this command against a database with limbo transactions, Progress displays a message similar to the following:

```
Commit transaction 760, on coordinator sports1 #768 (y to commit/n to
abort)? (2039)
```

If you type **y**, Progress commits the transaction. If you type **n**, Progress aborts the transaction.

PROUTIL BIGROW Qualifier

Specifies the number of BI clusters to preformat for a database.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C bigrow` *n* **[** `-r` **]**

db-name

Specifies the database where you want to create BI clusters.

n

Specifies the number of BI clusters to create for the specified database. Note that Progress creates 4 BI clusters at startup. If you specify a BIGROW value of 5, then you will have a total of 9 BI clusters.

`-r`

Nonreliable I/O startup parameter; turns off reliability.

NOTES

- You can calculate the number of BI clusters for a database by dividing the BI file physical size by the BI cluster size. For example, a database BI file with a BI cluster size of 128K and a physical size of 917,504 has 7 BI clusters.
- By default, Progress creates 4 BI clusters at startup.
- The database must be offline to use the BIGROW qualifier.
- Use the BIGROW qualifier immediately after truncating the BI file.
- See Chapter 14, "Managing Performance," for information about using the BIGROW qualifier.

PROUTIL BULKLOAD Qualifier

Loads data files into a database.

NOTE: The bulk loader works only with Progress databases. Some non-Progress databases offer a similar bulk-loading tool. If such a tool is not available, use the standard load option in the Progress Data Administration tool or Data Dictionary.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* [`-yy` *n*] `-C BULKLOAD` *fd-file* [`-B`*n*]

db-name

Specifies the database you are using.

`-yy` *n*

Century Year Offsett startup parameter; *n* specifies a four-digit year (1900, for example) that determines the start of a 100-year period in which any two-digit DATE value is defined. The default is 1950, but the `-yy` *n* value must match the `-yy` *n* value used when the data was dumped.

fd-file

Identifies the bulk loader description file you are using.

`-B`*n*

Blocks in DataBase Buffers startup parameter; *n* specifies the number of blocks.

NOTE

You must create the bulk loader description file and load the data definitions (.df) files for the database before you can run the Bulk Loader utility. See Chapter 13, "Dumping and Loading," for information about creating the bulk loader description file and loading data definitions files.

PROUTIL BUSY Qualifier

Determines whether the database is currently in use. This information is useful before performing a database backup or shutting down a database. A database is in use if it is being used in single-user mode, multi-user mode, or by a Progress utility program.

To get the information, you must run PROUTIL with the BUSY qualifier and test the command return code in a UNIX script or Windows batch file.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C busy`

db-name

Specifies the database you are using.

NOTE

- When the PROUTIL command completes, it returns a code you can test in a UNIX script or Windows batch file. The return codes for the BUSY qualifier are shown in the following table:

SYNTAX

Operating System	Return Code	Description
UNIX Windows	0	Database is not in use.
	6	Database is in use.
	64	Database is in process of starting up.

EXAMPLE

- This example shows how you might use the BUSY qualifier on UNIX in a script that tests whether the database is busy:

```
proutil mydb -C busy

if [ $? != 0 ]
then
  echo
  echo "Do you want to use ‘proshut’ to force users off\
    the system?"
  read ans
  if [ "$ans" = y ]
  then
    proshut -by mydb
  else
    echo "Backup will not be performed."
    exit
  fi
fi
echo "Beginning backup."

# Backup procedure
```

PROUTIL CHANALYS Qualifier

Displays information about free and RM chains. It calculates the number of blocks found both in the free chain and the RM chain.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C chanalys`

db-name

Specifies the database you are using.

PROUTIL CODEPAGE-COMPILER Qualifier

Compiles a conversion map file from text format to binary format. A *conversion map file* is a code page description file that contains various tables of information and uses a specific format. For more information about conversion map files, see the *Progress Internationalization Guide*.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil -C codepage-compiler` *`inputfile outputfile`*

inputfile

Specifies the name of the text conversion map file. This file must follow a specific format.

outputfile

Specifies the name of the binary conversion map file.

PROUTIL CONV89 Qualifier

Converts a multi-volume Version 8 database to a Version 9 database. To convert a database use the following command:

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C conv89`

db-name

Specifies the source database name.

NOTES

- You can only use the CONV89 qualifier to convert a multi-volume Version 8 database to Version 9. If you want to convert a single-volume Version 8 database to Version 9, you must first convert it to a multi-volume Version 8 database. For more information, see Chapter 4, “Creating and Deleting Databases.”
- You must truncate the BI file before using PROUTIL CONV89.
- You should disable after-imaging and two-phase commit before starting the conversion; however, if you forget to do so, PROUTIL will disable after-imaging and two-phase commit for you. PROUTIL issues an informational message when after-imaging and/or two-phase commit is disabled.
- If you intend to enable two-phase commit, add a transaction log area to your Version 9 database. For a description of how to add storage areas to your database, see the “Progress Structure Add Utility” section in Chapter 9, “Maintaining Database Structure.”

PROUTIL CONVCHAR Qualifier

Converts a database's character set or identifies a database's character set (for undefined databases). When converting a database's character set, PROUTIL CONVCHAR converts all of the textual data in the database.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C convchar` [`analyze` | `charscan` | `convert`] [*codepage*] [*character-list*]

db-name

Specifies the database you are converting.

`analyze`

Scans the database and displays the fields that would be converted using the `convert` function.

`charscan`

Searches every character field for the occurrence of any character from the provided character list and reports the table name, field name, record ID of a match, and the total number of occurrences. In addition, PROUTIL CONVCHAR CHARSCAN performs the same analysis that PROUTIL CONVCHAR ANALYSIS performs.

If invalid data are entered with the `charscan` option, PROUTIL CONVCHAR generates an error message and continues with the scan.

`convert`

Converts a database's character data to the target code page and labels the database.

codepage

Specifies the value of any single-byte, double-byte, or Unicode codepage. Possible values are undefined or any (target) code page name for a conversion table in your conversion map file (by default, DLC/convmap.cp). If you specify a code page name, the PROUTIL CONVCHAR utility must find the appropriate conversion table in the conversion map file. If you specify undefined, no conversions take place and the database's code page name is changed to undefined.

If you specify a code page name against a database with a code page name of undefined, no conversions take place and the database's code page name is changed to the value you specify.

NOTE: When you change the code page of a database from "undefined" using `proutil` *dbname* `-C convchar convert` *codepage* PROUTIL CONVCHAR generates warning messages. After you receive these messages, you might need to load the collation table for the code page. The collation table data definition files have a .df extension and are located in $DLC/prolang/<*language*>.

When you specify a codepage value with `analyze`, it scans the database and identifies the database's current code page encoding and the fields that require translation if you were to use the convert function with that code page specification.

When you use the codepage parameter with `convert`, it specifies the code page to which you want to convert the database and specifies the code page name for the database.

character-list

Specifies a quoted string of comma-separated numbers in either hex or decimal. These numbers represent character values in the codepage. Specify ranges with a minus sign ("`128 - 144`").

NOTE: Quotes are not needed if there are no blanks within the list (`128-144,122`).

Hex values must begin with `0x`. For example, the syntax to run PROUTIL CONVCHAR with the decimal list "`128, 129, 130`" provided in hex is:

```
proutil sports2000 -C convchar charscan ibm850 "0x80, 0x81, 0x82"
```

Note also that hex and decimal values may be mixed. For example:

```
proutil sports2000 -C convchar charscan 1253 "128, 0xC2, 0x7f, 122"
```

If a range contains valid and invalid characters, PROUTIL CONVCHAR ignores the invalid character. For example, this syntax

```
proutil sports2000 -C convchar charscan 1253 "49854 - 50050"
```

searches and returns the following:

```
Charscan searching for utf-8 character: 49854 0xc2be. (6570)
Charscan searching for utf-8 character: 49855 0xc2bf. (6570)
Charscan searching for utf-8 character: 50048 0xc380. (6570)
Charscan searching for utf-8 character: 50049 0xc381. (6570)
Charscan searching for utf-8 character: 50050 0xc382. (6570)
```

If `charscan` is selected but no character list is specified, the `analyze` function performs.

For more information about character set processing, see the *Progress Internationalization Guide*.

PROUTIL CONVFILE Qualifier

Converts a text file from one character set to any other character set.

SYNTAX

Operating System	Syntax
UNIX Windows	proutil -C convfile { *file-name* convert using *table-name* | *file-name* [analyze]}

file-name

Specifies the name of the file you are converting or analyzing.

convert using *table-name*

Specifies the name of the file containing the conversion table. This file requires the following format:

```
# optional comment lines begin with the # character
SOURCE source-codepage-name
TARGET target-codepage-name
/*000-015*/000  001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
/*016-031*/016  017 018 019 020 021 022 023 024 025 026 027 028 029 030 031
/*032-047*/032  033 034 035 036 037 038 039 040 041 042 043 044 045 046 047
                                          .
                                          .
                                          .
/*240-255*/240  241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
```

where *source-codepage-name* is the name of the character set of *file-name*, and *target-codepage-name* is the name of the desired character set for *file-name*.

The conversion table is composed of 256 cells, numbering from 0 to 255. You can build your own conversion table, or you can use default tables stored in the `DLC/prolang` directory if appropriate. The tables in `DLC/prolang` are as follows:

- **1250-852.dat** — Converts from code page 1250 to IBM code page 852.
- **1250-il2.dat** — Converts from code page 1250 to iso8859-2.
- **1254-857.dat** —Converts from IBM code page 1254 to code page 857.
- **852-1250.dat** — Converts from IBM code page 852 to code page 1250.
- **852-il2.dat** — Converts from IBM code page 852 to iso8859-2.
- **857-1254.dat** — Converts from IBM code page 857 to code page 1254.
- **cn850.dat** — Converts iso8859-1 to IBM code page 850.
- **cn8859-1.dat** — Converts IBM code page 850 to iso8859-1.
- **cnnull.dat** — Performs no conversion; you might want to use this file as a template file for creating new conversion tables.
- **il2-1250.dat** — Converts from iso8859-2 to code page 1250.
- **il2-852.dat** — Converts from iso8859-2 to IBM code page 852.

If you create your own conversion table, you must understand how PROUTIL CONVFILE uses the table. For each character in *file-name*, PROUTIL CONVFILE uses the character's numeric value to index it against the table. When it locates the corresponding cell, PROUTIL CONVFILE replaces the character in the text file with the character value in that cell. Therefore, you must make a copy of the text file before you run the PROUTIL CONVFILE utility, because if the utility does not completely convert the file, the data in it is corrupted.

`analyze`

Displays the number of occurrences of each character in *file-name*. You might be able to determine the character set of *file-name* by comparing the output of this option against different code pages.

PROUTIL DBANALYS Qualifier

Displays statistical information about index, record, and free chain blocks.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C dbanalys`

db-name

Specifies the database you are using.

EXAMPLE

The following is a sample display of PROUTIL DBANALYS output:

```
SUMMARY FOR AREA "Inventory" : 8
------------------------------------------------------------
                     Records            Indexes                Combined
NAME                Size   Tot %       Size   Tot %             Size  Tot %
PUB.Bin             26.3K    1.1       16.4K    0.7            42.8K    1.8
PUB.InventoryTrans  3661B    0.1          0B    0.0            3661B    0.1
PUB.Item            7662B    0.3       8117B    0.3           15779B    0.6
PUB.POLine         217.5K    9.1       51.9K    2.2           269.4K   11.3
PUB.PurchaseOrder   68.6K    2.9       19.0K    0.8            87.6K    3.7
PUB.Supplier        1177B    0.0          97B    0.0            1274B    0.1
PUB.SupplierItemXr  1118B    0.0       1166B    0.0            2284B    0.1
PUB.Warehouse       1288B    0.1        440B    0.0            1728B    0.1

Size key:
 B = bytes
 K = kilobytes
 M = megabytes
 G = gigabytes
```

PROUTIL DBAUTHKEY Qualifier

Sets an authorization key for a database. When you compile source code, Progress includes the value of this key in your r-code. CRC-based r-code that does not include the correct authorization key will not run against your database.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C dbauthkey` *old-key new-key*

db-name

Specifies the database you are using.

old-key

Specifies the old authorization key.

new-key

Specifies the new authorization key.

NOTES

- You use the DBAUTHKEY qualifier to set, change, and remove authorization keys. The following table lists the values you must enter for each task:

Task	*old-key* Value	*new-key* Value
Set the authorization key	Plus sign (+)	Authorization key
Change the authorization key	Current authorization key	New authorization key
Remove the authorization key	Current authorization key	Plus sign (+)

- Once you set the authorization key, do not forget it. You cannot change or remove the authorization key without knowing its current value.
- After you set, change, or remove the authorization key for a database, you can use the RCODEKEY qualifier to update your r-code without recompiling it.

PROUTIL DBIPCS Qualifier

Displays status information for shared-memory segments attached by all Progress databases on the system. This qualifier is valid only on UNIX shared-memory systems.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil -C dbipcs`

EXAMPLE

The following is a sample display of PROUTIL DBIPCS output:

```
PROGRESS SHARED MEMORY STATUS
 ID  ShMemVer  Seg#  InUse   Database
 68     -       -     -      (not PROGRESS)
100     3       0     Yes    /db/work5/sports
101     3       1     -      /db/work5/sports
120     3       0     No     /db/work5/test
```

DISPLAY FIELDS

`ID`

Indicates the shared-memory ID.

`ShMemVer`

Specifies the shared-memory version.

`Seg#`

Indicates the shared-memory segment number. One database can own more than one segment.

`InUse`

Specifies whether the segment is in use. Yes or No values are displayed only if the segment is number 0. All other segments show a hyphen (-). To determine whether a set of segments is in use, check the InUse value of segment 0 for the relevant database.

`Database`

Represents the full pathname of the database.

NOTES

- The PROMON Activity screen shows the amount of shared memory used by Progress and the number of shared-memory segments.
- On UNIX, if the broker dies or is killed by means other than the database shutdown command, it does not release shared-memory segments. Use PROUTIL DBIPICS to verify whether shared memory is frozen. Use the UNIX command `ipcrm -m` to free it.
- On UNIX, use some `ipcs -m` to determine the size and number of shared-memory segments in use. Subtract from SHMALL to see how many shared-memory segments are left. If the broker dies or is killed by means other than the database shutdown command, it does not release attached shared-memory segments. Use PROUTIL -C DBIPCS to verify shared memory is frozen; use the UNIX command `ipcrm -m` to free it.

PROUTIL DUMP Qualifier

Performs a binary dump of a database table to a file. PROUTIL DUMP writes data from a table to a dump file. When the procedure finishes, it reports the number of records written to the dump file.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C dump [` *owner-name.*`]` *table-name directory* `[ -index num ]`

db-name

Specifies the database where the dump will occur. If the database is not within the current working directory, you need to define the complete path.

owner-name

Specifies the owner of the table containing the data you want to dump. You must specify an owner name unless the table's name is unique within the database, or the table is owned by "PUB." By default, Progress 4GL tables are owned by PUB.

table-name

Specifies the name of the table containing the data you want to dump.

directory

Specifies the name of the target directory where the data will be dumped.

`-index num`

Specifies the index to use to dump the table's contents. If you choose not to use this option, the command uses the primary index to dump the table.

NOTES

- See Chapter 13, "Dumping and Loading," for more information about the DUMP qualifier.
- The PROUTIL DUMP and LOAD utilities use cyclic redundancy check (CRC) values to establish the criteria for loading.

  The Progress database provides a flexible storage architecture and the ability to relocate objects, such as tables and indexes, while the database remains online. As a result, when you perform a binary load operation, the table numbers in a binary dump file might not match the table numbers in the target database. Therefore, when you perform a binary load operation, the criteria for loading tables is based solely on cyclic redundancy check (CRC) values, and not table numbers.

  For example, when you dump a table, the PROUTIL utility calculates a CRC value for the table and stores it in the header of the binary dump file. When you load the table, PROUTIL matches the CRC value stored in the header with the CRC value of the target table. The values must match or the load is rejected.

  You can load a binary dump file created with a previous version of the PROUTIL DUMP utility, because the current version of PROUTIL LOAD uses the CRC value established when the file was originally dumped. Consequently, the Progress database maintains backwards compatibility.

  However, you cannot use a previous version (Version 8.3 or earlier) of the PROUTIL LOAD utility to load a binary dump file created using the current version (version 9.0 or later) of the PROUTIL DUMP utility. The previous versions of PROUTIL DUMP and LOAD did not use CRC values to establish the criteria for loading, but instead used other mechanisms, such as:

  – Looking up table RecIDs in a target database using the table number stored in the header of the binary dump file.
  – Matching table numbers in the header of the binary dump file with table numbers in a target database.
  – Comparing the number of fields in the binary dump file with the number of fields in the target database.

- PROUTIL DUMP writes data from a table to a dump file. The name of the resulting dump file depends on the owner of the table. By default, Progress 4GL tables are owned by "PUB." When tables owned by PUB are dumped to a file, the filename is the table name with `.bd` appended. For example, *`tablename`*`.bd`.

  However, when tables owned by anyone other than PUB are dumped to a file, the resulting filename contains the owner name and table name. For example, *`ownername_tablename`*`.bd`

- On UNIX systems that have a 2GB-file-size limitation (Alpha OSF does not), PROUTIL DUMP creates multiple files when you dump a table larger than 2GB. For example, when you dump data from a table with the name "customer" that is 6.4GB, PROUTIL DUMP creates four binary dump files: `customer.bd, customer.bd2,` and `customer.bd3,` each of which is approximately 2GB, and `customer.bd4,` which is approximately 0.4GB. The PROUTIL DUMP procedure adds header blocks to the binary dump files. As a result, the total size of the binary dump files is slightly larger than the table itself.

  On Windows NT and Alpha OSF, however, there is no 2GB-file-size limitation. On Windows NT and Alpha OSF, PROUTIL DUMP creates only one binary dump file regardless of the size of the table.

- PROUTIL DUMP supports dumping binary large objects (BLOBS).

PROUTIL DUMPSPECIFIED

Performs a binary dump of a database field to a file.

SYNTAX

Operating System	Syntax
UNIX Windows	proutil *db-name* -C dumpspecified [*owner-name.*] *table-name.field-name operator field-value directory*

db-name

Specifies the database where the dump will occur. You must completely define the path.

owner-name

Specifies the owner of the table containing the data you want to dump. You must specify an owner name unless the table's name is unique within the database, or the table is owned by "PUB." By default, Progress 4GL tables are owned by PUB.

table-name

Specifies the name of the table containing the data you want to dump.

field-name

Specifies the name of the field containing the data you want to dump.

operator

Specifies the operator to be used: EQ (equals to), GT (greater than), LT (less than), GE (greater than or equal to), or LE (less than or equal to).

field-value

Specifies the value against which the filed contents will be compared.

directory

Specifies the name of the target directory where the data will be dumped. A directory must be specified.

NOTES

- If a field needs to be compared to a string containing other than alpha characters (including spaces or numbers), single quote the string. For example: `'John Smith'`, `'11/14/01'`, or `'25 Main St'`.
- If specifying a monetary value, do not include the money unit. For example, instead of $5.50, enter `5.5`.

PROUTIL ENABLELARGEFILES Qualifier

Enables large file processing for a database.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C EnableLargeFiles`

db-name

Name of the database to be enabled for large files.

NOTES

- The database must be offline when enabling large file processing.
- There is no corresponding command to disable large file processing for a database.
- Once large file processing is enabled for a database, the database cannot be accessed by Progress Versions prior to 9.1C.

PROUTIL ENABLEPDR Qualifier

Enables the database to work with Peer Direct for database replication.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C enablepdr`

db-name

Specifies the name of database to be replicated.

NOTES

- Before enabling Peer Direct, you must grant Progress SQL-92 resource access to your Peer Direct administrator.
- Tables to be replicated must have a unique primary key.

PROUTIL HOLDER Qualifier

Determines whether the database currently in use is in single-user mode, multi-user mode, or in use by a Progress utility. This information is useful before performing a database backup or shutting down a database.

When you execute this command, PROUTIL returns a different return code depending on how you are running Progress. To get the information, you must run PROUTIL with the HOLDER qualifier and test the command return code in a UNIX script or Windows batch file.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C holder`

db-name

Specifies the database you are using.

NOTES

- When the HOLDER qualifier completes, it sets a return code that you can test in a UNIX script or Windows batch file. The return codes for the HOLDER qualifier on UNIX are shown in the following table:

SYNTAX

Operating System	Return Code	Description
UNIX Windows	0	Database not in use.
	14	Database locked by single user, PROUTIL, or RFUTIL.
	16	Database open in multi-user mode.

Any other nonzero return code indicates that an error has occurred.

- Return codes can be added or changed from one release to the next. Use scripts that depend on a specific return code with caution.
- This example shows how you might use the HOLDER qualifier on UNIX in a script that performs a database backup:

```
proutil mydb -C holder
retcode=$?  # this saves the return code
case $retcode in
0) echo "The database is not busy, ok to backup"
   ;;
14) echo "The database is locked, no backup done"
   exit $retcode
   ;;
16) echo "The database is busy in multi-user mode, no backup done"
   exit $retcode
   ;;
*) proutil failed, no backup done
   echo error code = $retcode
   exit $retcode
   ;;
esac
proutil mydb -C truncate bi
<<test the return value from proutil here>>
<<put the backup command here>>
rfutil mydb -C mark backedup
```

PROUTIL IDXANALYS Qualifier

Displays information on index blocks. It calculates the number of index blocks and the percentage of the database used by the index blocks.

NOTE: The _UserStatus virtual system table displays the utility's progress. For more information see Chapter 14, "Managing Performance."

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C idxanalys`

db-name

Specifies the database to analyze.

EXAMPLE

The following is a sample display of PROUTIL IDXANALYS output:

```
INDEX BLOCK SUMMARY FOR AREA "Order" : 11
------------------------------------------------------
Table                  Index  Fields Levels  Blocks   Size   % Util Factor
PUB.BillTo
  custnumbillto          46       2      1       1     27B      2.7    1.0
PUB.Order
  CustOrder              48       2      2      44   39.1K     91.4    1.5
  OrderDate              49       1      2      33   18.5K     57.7    2.1
  OrderNum               47       1      2      38   35.3K     95.5    1.5
  OrderStatus            50       1      2       7   3808B     54.6    2.1
  SalesRep               51       1      2       9   6226B     69.5    1.9
PUB.OrderLine
  itemnum                23       1      2      56   36.6K     67.2    1.9
  orderline              22       2      3     139  132.7K     98.2    1.0
  OrderLineStatus        24       1      2      10   7354B     73.8    1.8
PUB.ShipTo
  custnumshipto          45       2      1       1     36B      3.6    1.0
```

PROUTIL IDXBUILD Qualifier

Packs or consolidates index records to use disk space as efficiently as possible. It also:

- Compresses index blocks to minimize space usage.
- Activates deactivated indexes in the database.
- Repairs corrupted indexes in the database. (Index corruption is typically signaled by error messages.)

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C idxbuild` [`all`] [`-T` *dir-name*] [`-TB` *blocksize*] [`-TM` *n*] [`-B` *n*]

db-name

Specifies the database you are using.

`all`

Specifies that you want to rebuild all your indexes. PROUTIL automatically rebuilds all your indexes without asking about disk space requirements.

If you do not specify all, the following menu appears:

```
Index Rebuild Utility
-----------------------
Select one of the following
All  - Rebuild all of the indexes
Some - Rebuild only some of the indexes
Quit - Quit, do not rebuild

Enter your selection:
```

The following table describes the options:

Option	Action
All	Prompts you to verify whether you have enough disk space for index sorting.
Some	Prompts you for the indexes you want to rebuild, then prompts you to verify whether you have enough disk space for index sorting.
Quit	Quits without rebuilding any indexes.

NOTES

- Use IDXFIX to repair and IDXCOMPACT to compress online.
- Use the Temporary Directory (-T) startup parameter to identify or redirect temporary files created by the PROUTIL utility to a specified directory when sorting and handling space issues.
- Use the Speed Sort (-TB), Merge Number (-TM), and Blocks in Database Buffers (-B) startup parameters to improve index rebuild performance. For more information on these parameters, see the *Progress Startup Command and Parameter Reference*.
- IDXBUILD does not repair corrupted data.
- Use the following formulas to determine whether you have enough free space to sort the indexes:
  - If you rebuild all the indexes in your database, sorting the indexes requires free space that can be as much as 75 percent of the total database size.
  - If you rebuild an individual index, sorting that index requires free space that can be the size of one index entry * the number of records in the file * 2.

- The Index Rebuild utility rebuilds an index or set of indexes in three phases:
  - The utility scans the database, clearing all index blocks that belong to the indexes and rebuilds and adds those blocks to the free-block list.
  - The utility scans the database file and rebuilds all the index entries for every data record. If you chose to sort the index, the utility writes the index entries to the sort file. Otherwise, the utility writes the index entries to the appropriate index at this point.
  - If you indicated that you have enough space to sort the indexes, the utility sorts the index entries in the sort file into groups and enters those entries in one index at a time.
- For more information about using the IDXBUILD qualifier, see Chapter 14, “Managing Performance.”

PROUTIL IDXCHECK Qualifier

Checks Progress database indexes to determine whether an index is corrupt, and if it is, diagnoses the problem. This lets you know whether you have to perform an index rebuild before actually running PROUTIL IDXBUILD.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C idxcheck [ all ]`

db-name

Specifies the database whose index you are checking.

`all`

Specifies that you want to check all your indexes.

If you do not specify all, the following menu appears:

```
Index Check Utility
---------------------------
Select one of the following:
All - Check all the indexes
Some - Check only some of the indexes
Quit - Quit, do not check

Enter your selection:
```

The following table describes the options:

Option	Action
All	Checks all the indexes.
Some	Prompts you for the indexes you want to check.
Quit	Quits without checking any indexes.

PROUTIL IDXCHECK performs the following operations for each index it checks:

- Reads the contents of the index and the contents of the file, verifies that all the values in the records are indexed, and verifies that each value in the index is in the associated record.
- Performs various checks on the data structures in the index to verify that there is no corruption.

IF PROUTIL IDXCHECK completes successfully, it ensures that all FIND, CAN-FIND, GET, FOR EACH, and PRESELECT statements that use those indexes work correctly. If errors result, the problem index might produce unexpected results with those statements.

NOTES

- When PROUTIL IDXCHECK finds any corruption, it displays error messages. If error messages appear, save the database and the output messages for analysis by Progress Software Corporation, then run PROUTIL IDXBUILD.
- IDXCHECK displays error and warning messages on the screen and logs them in the log file. It also displays and logs a success or failure indication, along with the number of errors and warnings issued.
- IDXCHECK might also display warning messages. Although these messages indicate that some condition is a problem, the index is still valid. Check the log file for details.

PROUTIL IDXCOMPACT Qualifier

Performs index compaction online. This is recommended when the PROUTIL IDXANALYS utility indicates that space utilization of an index is reduced to 60% or less. Index compaction increases space utilization of the index block to the compacting percentage specified by *n*.

SYNTAX

Operating System	Syntax
UNIX Windows	proutil *db-name* -C idxcompact [*owner-name.*] *table-name.index-name* [*n*]

db-name

Specifies the source database name.

owner-name

Specifies the owner of the table containing the data you want to dump. You must specify an owner name unless the table's name is unique within the database, or the table is owned by "PUB." By default, Progress 4GL tables are owned by PUB.

table-name

Specifies the source table containing the source index to be compacted.

index-name

Specifies the source index to be compacted.

n

Specifies the degree of index compaction. You can specify an integer >=50 and <=100. The default value is 80. If you do not specify *n*, 80 is used.

NOTES

- Performing index compaction reduces the number of blocks in the B-tree and possibly the number of B-tree levels, which improves query performance.
- The index compacting utility operates in phases:
  - Phase 1: If the index is a unique index, the delete chain is scanned and the index blocks are cleaned up by removing deleted entries.
  - Phase 2: The nonleaf levels of the B-tree are compacted starting at the root working toward the leaf level.
  - Phase 3: The leaf level is compacted.
- PROUTIL IDXCOMPACT may be run either online or offline.
- In addition to compacting an index, this utility clears dead entries left after entries have been deleted from unique indexes.
- Because index compacting is performed online, other users can use the index simultaneously for read or write operation with no restrictions. Index compacting only locks 1 to 3 index blocks at a time, for a short time. This allows full concurrency.
- The IDXCOMPACT utility does not lock any record or table.
- No other administrative operation on the index is allowed during the compacting process.
- In rare cases where the required percentage of compaction is very high, the compacting percentage might not be reached. Repeating the compacting process a second time might obtain better results.
- See Chapter 9, “Maintaining Database Structure,” for a description of how to monitor the progress of this utility using the _UserStatus Virtual System Table (VST).

PROUTIL IDXFIX Qualifier

Checks the Progress database records and indexes to determine whether an index is corrupt or a record has a missing or incorrect index. You can specify whether to scan the database, the indexes, or both. You can run IDXFIX online or offline.

NOTE: The _UserStatus virtual system table displays the utility's progress. For more information, see Chapter 9, "Maintaining Database Structure."

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-silent -C idxfix`

db-name

Specifies the database whose index you are checking.

`silent`

Indicates that repetitive messages are not sent to the screen or the log file.

The Index Fix utility displays the following menu:

```
                Index Fix Utility

          Select one of the following:
          1. Scan records for missing index entries.
          2. Scan indexes for invalid index entries.
          3. Both 1 and 2 above.
          4. Cross-reference check of multiple indexes for a table.
          5. Build indexes from existing indexes.
          6. Delete one record and its index entries.
          7. Quit

          Enter your selection:
```

The following table describes the options:

Option	Action
1	Scans the database records for missing or incorrect indexes.
2	Scans the index for corrupt index entries. You can specify whether to scan all indexes or a specific set of indexes.
3	Checks the index entries, then checks the database entries.
4	Prompts you for the table and indexes for which you want to run a cross-reference check. Allows you to choose two or more indexes from one table. It first scans them for invalid keys, then scans them for invalid records.
5	Prompts you to specify the table and the index that you want to use as the source from which to build the index.
6	Prompts you to specify the recid of the record you want to delete. Deletes one record and all its indexes from the database. Use this option when a record has damaged indexes.
7	Ends the PROUTIL Index Fix utility.

NOTES

- PROUTIL IDXFIX performs the following operations for each index it checks:
  - Reads the contents of the index and the contents of the file, verifies that all the values in the records are indexed, and verifies that each value in the index is in the associated record.
  - Performs various checks on the data structures in the index to verify that there is no corruption.
- IDXFIX displays error messages on the screen and logs them in the log file. It also displays and logs a success or failure indication, along with the number of errors and warnings issued. IDXFIX might also display warning messages. Although these messages indicate that some condition is a problem, the index is still valid. Check the log file for details.

- Index Fix does not provide a comparison of the index scan and the database scan when you run them online because the database can change during operation.
- Index Fix is designed to wait if a record is in the process of being updated, thus ensuring that it does not incorrectly change a user action in process. However, because changes can occur during the scans, the reported data might not exactly match the database at completion. Index Fix displays this warning when you run it online.
- To run Index Fix online, it must be run as a self-service session.
- Index Fix does not delete or disable indexes, but when you run a full database scan with fix offline and it is successful, it enables a disabled index if no errors are found.
- Enabling indexes online is not advisable because it is not possible to detect changes that are being made to indexes while the process is running.
- While IDXFIX can ensure that an index is complete and correct, it cannot make any improvement to the index's utilization level.
- See Chapter 9, "Maintaining Database Structure," for a description of how to monitor the progress of this utility using the _UserStatus Virtual System Table (VST).

PROUTIL IDXMOVE Qualifier

Moves an index from one application data area to another while the database remains online. You might be able to improve performance by moving indexes that are heavily used to an application data area on a faster disk.

The PROUTIL IDXMOVE utility operates in two phases:

- Phase 1: The new index is being constructed in the new area. The old index remains in the old area, and all users can continue to use the index for read operations.
- Phase 2: The old index is being killed, and all the blocks of the old index are being removed to the free block chain. For a large index, this phase might take a significant amount of time. During this phase all operations on the index are blocked until the new index is available; users accessing the index might experience a freeze in their applications.

NOTE: The _UserStatus virtual system table displays the utility's progress. For more information see Chapter 9, "Maintaining Database Structure."

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C idxmove` [*owner-name.*] *table-name.index-name area-name*

db-name

Specifies the name of the database containing the table.

owner-name

Specifies the owner of the table containing the data you want to dump. You must specify an owner name unless the table's name is unique within the database, or the table is owned by "PUB." By default, Progress 4GL tables are owned by PUB.

table-name

Specifies the source table containing the index to be moved.

index-name

Specifies the name of an index to move.

area-name

Specifies the area name of the target application data area into which the index is to be moved. Area names that contain spaces must be quoted. For example, “Area Name.”

NOTES

- While you can move indexes online, no writes to the table or its indexes are allowed during the move. The IDXMOVE utility acquires a SHARE lock on the table, which blocks all attempts to modify records in the table. Progress Software recommends that you run the utility during a period when the system is relatively idle or when users are doing work that does not access the table.
- No other administrative operation on the moved index will be allowed during the move of the index. It will be blocked. For example, you cannot run an index move utility and at the same time run the index fix or the index compacting utilities on the same index.
- Because the index move utility needs to acquire a share lock on the table, there is a possibility that it will have to wait before it can acquire the lock and start operating.

PROUTIL IOSTATS Qualifier

Provides current statistics for active databases. The statistics include buffered, unbuffered, and logical I/O database operations. The statistics are cumulative from database startup.

SYNTAX

Operating System	Syntax
UNIX Windows	proutil *db-name* -C iostats

db-name

Specifies the active database where you are running database I/O statistics.

EXAMPLE

The following is a sample display of PROUTIL IOSTATS output:

```
Database name is sports (2524)
FILE          BUFFERED        UNBUFFERED          LOGICAL (2556)
            Writes  Reads    Writes  Reads   Reads  Writes  Extends (2557)
================================================================================
sports.b1        0      0       317    389     388     252        2
sports.d1        1    173        11      0     171      11        0
sports_7.d1    773   6792         0      0    6790     772        0
sports_7.d2    270   1702        10      0    1701     269        4
sports_8.d1    649    715         0      0     713     648        0
sports_8.d2      1      1         0      0       0       0        0
sports_9.d1      1      2         0      0       0       0        0
sports_9.d2      1      1         0      0       0       0        0
sports_10.d1     1      2         0      0       0       0        0
sports_10.d2     1      1         0      0       0       0        0
```

DISPLAY FIELDS

FILE

Displays the name of the file where the statistics are displayed. The file types can include: database files (.db extensions), Before-image files (.bi extensions), After-image files (.ai extensions), and application data extents (.d*n* extensions).

BUFFERED

Displays the number of buffered reads and writes to the database file for the associated row.

UNBUFFERED

Displays the number of unbuffered reads and writes to the database file for the associated row.

LOGICAL

Displays the number of client requests for database read and write operations for the associated row.

NOTES

- The statistics are cumulative and are reset at database open.
- IOSTATS is available only on multi-user databases.
- IOSTATS provides a useful alternative to PROMON when you are only interested in statistics on your database extents.
- You can use IOSTATS to determine if your files are opened in buffered or unbuffered mode.

PROUTIL LOAD Qualifier

Performs a binary load of database contents.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C load` *filename* [`build indexes` [`-TB` *blocksize*] [`-TM` *n*] [`-T` *dir-name*] | [`-SS` *sort-file-directory-specification*]]

db-name

Specifies the database where you want to load the data. You must completely define the path.

filename

Specifies the binary dump file that you want to load. You must completely define the path.

`build indexes`

Indicates that PROUITL LOAD will simultaneously build the indexes and perform the load.

`-TB` *n*

Specifies that the index rebuild will be performed using Speed Sort. *n* indicates the allocated block size, in kilobytes. For more information on the Speed Sort (`-TB`) parameter, see the *Progress Startup Command and Parameter Reference*.

`-TM` *n*

Specifies the merge number. *n* indicates the number of blocks or streams to be merged during the sort process. For more information on the Merge Number (`-TM`) parameter, see the *Progress Startup Command and Parameter Reference*.

`-T` *dir-name*

Specifies the name of the directory in which the temporary files are stored. If you do not use this parameter, PROUTIL places temporary files in the current working directory. For more information about the Temporary Directory (`-T`) parameter, see the *Progress Startup Command and Parameter Reference*.

`-SS` *sort-file-directory-specification*

Identifies the location of a multi-volume sort file specification. If you use the Sort Directory Specification (`-SS`) parameter, PROUTIL does not use the Temporary Directory (`-T`) parameter.

NOTES

- See Chapter 13, "Dumping and Loading," for more information about the LOAD qualifier.
- The PROUTIL DUMP and LOAD utilities use cyclic redundancy check (CRC) values to establish the criteria for loading.

  The Progress database provides a flexible storage architecture and the ability to relocate objects, such as tables and indexes, while the database remains online. As a result, when you perform a binary load operation, the table numbers in a binary dump file might not match the table numbers in the target database. Therefore, when you perform a binary load operation, the criteria for loading tables is based solely on cyclic redundancy check (CRC) values, and not table numbers.

  For example, when you dump a table, the PROUTIL utility calculates a CRC value for the table and stores it in the header of the binary dump file. When you load the table, PROUTIL matches the CRC value stored in the header with the CRC value of the target table. The values must match or the load is rejected.

  You can load a binary dump file created with a previous version of the PROUTIL DUMP utility, because the current version of PROUTIL LOAD uses the CRC value established when the file was originally dumped. Consequently, the Progress database maintains backwards compatibility.

  However, you cannot use a previous version (Version 8.3 or earlier) of the PROUTIL LOAD utility to load a binary dump file created using the current version (Version 9.0 or later) of the PROUTIL DUMP utility. The previous versions of PROUTIL DUMP and LOAD did not use CRC values to establish the criteria for loading, but instead used other mechanisms, such as:

  – Looking up table RecIDs in a target database using the table number stored in the header of the binary dump file.
  – Matching table numbers in the header of the binary dump file with table numbers in a target database.
  – Comparing the number of fields in the binary dump file with the number of fields in the target database.

- On UNIX systems that have a 2GB file size limitation (Alpha OSF does not), PROUTIL DUMP creates multiple files when you dump a table larger than 2GB. For example, when you dump data from a table with the name "customer" that is 6.4GB, PROUTIL DUMP creates four binary dump files: `customer.bd`, `customer.bd2`, and `customer.bd3`, each of which is approximately 2GB, and `customer.bd4`, which is approximately 0.4GB. The PROUTIL DUMP procedure adds header blocks to the binary dump files. As a result, the total size of the binary dump files is slightly larger than the table itself.

  On Windows NT and Alpha OSF, however, there is no 2GB file size limitation. On Windows NT and Alpha OSF, PROUTIL DUMP creates only one binary dump file regardless of the size of the table.

  To load multiple binary dump files into the target database, specify each file individually. For example:

```
proutil db-name -C load customer.bd
proutil db-name -C load customer.bd2
proutil db-name -C load customer.bd3
proutil db-name -C load customer.bd4
```

- PROUTIL LOAD supports loading binary large objects (BLOBS).
- When using PROUTIL LOAD with the BUILD INDEXES qualifier, PROUTIL marks the existing indexes as inactive. Once PROUTIL successfully creates the indexes, it marks the indexes active. This means that if the binary load is aborted for any reason, PROUTIL leaves the table with no active indexes.
- Tables loaded with the `build indexes` qualifier must be empty.

PROUTIL MVSCH Qualifier

Frees disk space by moving schema after converting a database from Version 8 to Version 9.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C mvsch`

db-name

Specifies the name of the converted database.

NOTES

- After a database is converted from Version 8 to Version 9, the database's schema and data are located within the Schema Area (Area 6). After conversion, it is possible to move data into new areas by using PROUTIL dump and load or bulkload qualifiers, the Database Administration Tool, the Database Dictionary, or 4GL code. However, Area 6 continues to hold disk space, even after the removal of data, because the schema remains. Once the area's schema is removed by using PROUTIL MVSCH, Area 6 can be truncated.
- You must truncate the database's BI file before using PROUTIL MVSCH.
- Always backup the database before using PROUTIL MVSCH.

  CAUTION: PROUTIL with the mvsch qualifier is a non-recoverable utility. If the execution fails, you cannot connect to the database.

PROUTIL RCODEKEY Qualifier

Inserts the authorization key into existing CRC-based r-code.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C rcodekey` *old-key new-key files* **...**

db-name

Specifies the database you are using.

old-key

Specifies the old authorization key.

new-key

Specifies the new authorization key.

files

Specifies the r-code files.

NOTES

- When you use the DBAUTHKEY qualifier to set, change, and remove authorization keys, you can use the RCODEKEY qualifier to update your r-code without compiling it. The following table lists the values you must enter for each task:

Task	*old-key* Value	*new-key* Value	Files
Set the authorization key	Plus sign (+)	Authorization key	R-code files
Change the authorization key	Current authorization key	New authorization key	R-code files
Remove the authorization key	Current authorization key	Plus sign (+)	R-code files

- Once you set the authorization key, do not forget it. You cannot change or remove the authorization key without knowing its current value.

PROUTIL TABANALYS Qualifier

Displays information about the degree of fragmentation for each table in a database. Also displays summary information about record sizes for each table.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C tabanalys`

db-name

Specifies the database to display information.

EXAMPLE

The following is a sample display of PROUTIL TABANALYS output:

```
RECORD BLOCK SUMMARY FOR AREA "Employee" : 7
------------------------------------------------------------
                                -Record Size (B)- ---Fragments--- Scatter
Table         Records       Size   Min   Max   Mean   Count   Factor   Factor
PUB.Benefits     21        848B    39    41     40      21      1.0      2.0
PUB.Department    7        211B    26    35     30       7      1.0      1.0
PUB.Employee     55       6434B    99   135    115      55      1.0      1.0
PUB.Family       72       3195B    38    51     44      72      1.0      1.4
PUB.Timesheet    25       1081B    42    45     43      25      1.0      1.0
PUB.Vacation     12        288B    24    24     24      12      1.0      1.5
```

DISPLAY FIELDS

`Table`

Table owner and table name.

`Records`

Total number of records in the database for the table.

`Bytes`

Total number of bytes used in the database for the table.

`Min`

Minimum number of bytes used by any record for the table.

`Max`

Maximum number of bytes used by any record for the table.

`Mean`

Mean number of bytes used by any record for the table.

`Count`

Total number of record fragments found in the database for the table.

`Factor`

Degree of record fragmentation for the table. This value is determined by the number of fragments divided by the ideal number of fragments (for example, if the data were freshly loaded). A value of 1.0 is ideal. A value of 2.0 indicates that there are twice as many fragments as there would be if the data were freshly loaded.

Use the Index value to determine whether to dump and reload your data to improve fragmentation. If the value is 2.0 or greater, dumping and reloading will improve performance. If the value is less than 1.5, dumping and reloading is not warranted.

`Scatter Factor`

Degree of distance between records in the table.

The best achievable Scatter Index value is that of a freshly loaded database. This is the baseline number against which you should compare future Scatter Index measurements. A value of 1 indicates that the records occupy completely contiguous database blocks. A value of 2 indicates that the records are scattered 10 times wider than the ideal.

Use this value to determine the performance impact caused by fragmentation. If the value is 1.5 or greater, performance will be poor for sequential record access, and dumping and loading the data might be warranted. A value of 1.5 or greater might also reduce performance for random access; however, this depends on the system type and the application.

NOTES

- The PROUTIL DBANALYS display includes the information displayed by PROUTIL TABANALYS.
- See Chapter 14, “Managing Performance,” for more information about database fragmentation.

PROUTIL TABLEMOVE Qualifier

Moves a table and optionally its associated indexes from one storage area to another while the database remains online.

NOTE: The _UserStatus virtual system table displays the utility's progress. For more information see Chapter 9, "Maintaining Database Structure."

SYNTAX

Operating System	Syntax
UNIX Windows	proutil *db-name* -C tablemove [*owner-name.*] *table-name table-area* [*index-area*]

db-name

Specifies the name of the database containing the table.

owner-name

Specifies the owner of the table containing the data you want to dump. You must specify an owner name unless the table's name is unique within the database, or the table is owned by "PUB." By default, Progress 4GL tables are owned by PUB.

table-name

Specifies the name of the table to be moved.

table-area

Specifies the area name of the target application data area into which the table is to be moved. Area names with spaces in the name must be quoted, for example, "Area Name."

index-area

Optionally, specifies the name of the target index area. If the target index area is supplied, the indexes will be moved to that area. Otherwise they will be left in their existing location. You can move indexes to an area other than the area to which the table is being moved. Area names with spaces in the name must be quoted, for example, "Area Name".

NOTES

- If you omit the *index-area* parameter the indexes associated with the table will not be moved.
- Moving the records of a table from one area to another invalidates all the ROWIDs and indexes of the table. Therefore, the indexes are rebuilt automatically by the utility whether you move them or not. You can move the indexes to an application data area other than the one to which you are moving the table. If you want to move only the indexes of a table to a separate application data area, use the PROUTIL IDXMOVE utility.

  Moving a table's indexes with the TABLEMOVE qualifier is more efficient than moving a table separately and then moving the indexes with the IDXMOVE utility. Moving a table separately from its indexes wastes more disk space and causes the indexes to be rebuilt twice, which also takes longer.

  The PROUTIL TABLEMOVE utility operates in phases:

  – Phase 1: The records are moved to the new area and a new primary key is built.

  – Phase 2: All the secondary indexes are built:

    If you did not specify the *index-area* parameter, then the indexes are rebuilt in their original area. If you did specify the *index-area* parameter, then all the indexes are moved to the new area where they are rebuilt.

  – Phase 3: All the records in the old area are removed.

  – Phase 4: All the old indexes are killed and the _StorageObject records of the indexes and the table are updated.

  – The _UserStatus virtual system table displays the utility's progress. For more information, see Chapter 20, "Virtual System Tables."

- Although PROUTIL TABLEMOVE operates in phases, it moves a table and its indexes in a single transaction. To allow a full recovery to occur when a transaction is interrupted, every move and delete of each individual record is logged. As a result, moving a table requires the BI Recovery Area to be several times larger than the combined size of the table and its indexes. Therefore, before moving your table, determine if your available disk capacity is sufficient to support a variable BI extent that might grow to more than three times the size of the table and its indexes.

- While you can move tables online, no access to the table or its indexes is recommended during the move. The utility acquires an EXCLUSIVE lock on the table while it is moving. An application that reads the table with an explicit NO-LOCK might be able to read records, but in some cases might get the wrong results, since the table move operation makes many changes to the indexes. Progress Software recommends that you run the utility during a period when the system is relatively idle, or when users are doing work that does not access the table.
- No other administrative operation on any index of the moved table is allowed during the table move.
- There is a possibility that the utility will have to wait for all the necessary locks to be available before it can start. This might take some time.

PROUTIL TRUNCATE AREA Qualifier

Deletes all the tables and indexes in the specified storage area.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C truncate area` *area-name*

db-name

Specifies the database that contains the application data storage areas that you want to truncate.

area-name

Specifies the name of the storage area you want to truncate. When you specify the area name, PROUTIL truncates the area even if it contains storage objects. If no area name is specified, PROUTIL truncates all areas not containing objects.

NOTES

- Use of this qualifier is an important step in removing application data storage areas and extents from a database.
- Deleting the contents of storage areas with this feature also allows for rapid dumping and loading. Use PROUTIL with the TRUNCATE AREA qualifier after dumping data, but before initiating the load.
- Before using PROUTIL with the TRUNCATE AREA qualifier, remove all tables and indexes from the application data storage areas you want to truncate.
- PROUTIL with the TRUNCATE AREA qualifier works by resetting the hi-water mark in the storage area back to the beginning of the storage area. This hi-water mark reset frees all of the space in the storage area for re-use. Any tables and indexes in the storage areas are initialized to the state they were in before they contained any rows or index entries. Before resetting the hi-water mark, the before image (.bi) file is truncated.
- To use this command, the database must be off-line and after imaging must be disabled.

- If the storage area does not contain any storage objects, then the command simply resets the hi-water mark. If the storage area does contain tables and or indexes, their names are listed and you must confirm to truncate the storage area.
- Indexes in other storage areas that are on tables in the storage area being truncated are marked as inactive.
- Empty index root blocks for indexes in the area being truncated are recreated.
- PROUTIL with the TRUNCATE AREA qualifier re-creates any template records in the new area.

NOTE: The schema area cannot be truncated.

For more information, see the "Progress Structure Remove Utility" section in Chapter 9, "Maintaining Database Structure."

PROUTIL TRUNCATE BI Qualifier

Performs three functions:

- Uses the information in the before-image (BI) files to bring the database and after-image (AI) files up to date, waits to verify that the information has been successfully written to the disk, then truncates the before-image file to its original length.
- Sets the BI cluster size using the Before-image Cluster Size (-bi) parameter.
- Sets the BI block size using the Before-image Block Size (-biblocksize) parameter.

SYNTAX

Operating System	Syntax
UNIX Windows	proutil *db-name* -C truncate bi **{ [** -G *n* **]** **|** -bi *size* **|** -biblocksize *size* **}**

db-name

Specifies the database you are using.

-G *n*

Specifies the number of seconds the TRUNCATE BI qualifier waits after bringing the database and AI files up to date and before truncating the BI file. The default wait period is 60 seconds. You might specify a shorter period for practice or test purposes. However, do not do so for any significant database, because a system crash could damage the database if the BI file is truncated before the writes to the database and AI files are flushed from the operating system buffer cache.

-bi *size*

Specifies the size of the cluster in kilobytes. The number must be a multiple of 16 ranging from 16 to 262,128 (16K to 256MB). The default cluster size is 524K. If you use a value that is not a multiple of 16, PROUTIL rounds the value up to the next multiple of 16.

-biblocksize *size*

Specifies the size of the BI blocks in each buffer in kilobytes. The valid values are 1, 2, 4, 8, and 16. The default -biblocksize is 8K. A value of 0 tells Progress to use the default block size. The block size cannot be smaller than the database block size.

NOTES

- The Before-image Block Size (`-biblocksize`) parameter changes the BI block size so that Progress reads and writes the blocks as one block.
- Use the PROSTRCT STATISTICS qualifier to display the block size for a database.
- If you change the BI block size or cluster size before backing up a database, when you restore the database, the blocks and clusters will be the size you specified before the backup.
- Progress reads all the BI blocks according to the size of the first block it reads. For example, if the first BI block is 8K, Progress reads and writes each block on an 8K boundary.
- For performance reasons, you might want to run PROUTIL with the `-C bigrow` qualifier to increase the number of BI clusters available to your database before starting your server.

PROUTIL UPDATEVST Qualifier

Loads the specified database with updated or new virtual system tables (VSTs). For a reference of the virtual system tables that Progress provides, see Chapter 20, "Virtual System Tables."

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C updatevst`

db-name

Specifies the database you are using.

NOTES

- The PROUTIL utility with UPDATEVST qualifier deletes all existing VST schema information, then re-creates all the VSTs from the most current information.

PROUTIL WBREAK-COMPILER Qualifier

Compiles a word-break table. For PROUTIL WBREAK-COMPILER to succeed, the word-break table source file must define a data structure named `word_attr`.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil -C wbreak-compiler` *`src-file rule-num`*

`src-file`

Identifies the word-break table source file to compile.

`rule-num`

Specifies an integer between 1 and 255 (inclusive) that will uniquely identify the compiled word-break table. PROUTIL WORD-BREAK COMPILER names the compiled version of the word-break table `proword.`*`rule-num`*. For example, if *`rule-num`* is 34, the name of the compiled word-break table file is `proword.34`.

NOTES

- See the *Progress Programming Handbook* for more information about word indexes and word-break table syntax.
- To reference your compiled word-break table (`proword.`*`rule-num`*), move it to the `$DLC` directory or set the PROWD environment variable to point to the location of the file. When you set the PROWD environment variable, append the value of *`rule-num`* to PROWD. For example:

```
PROWD34=proword.34; export PROWD34
```

- To apply the word-break rules to a database, use the WORD-RULES qualifier with the PROUTIL utility.

PROUTIL WORD-RULES Qualifier

Compiles a word rules file that tells Progress that the rules in the specified files apply to a database.

SYNTAX

Operating System	Syntax
UNIX Windows	`proutil` *db-name* `-C word-rules` *rule-numb*

db-name

Specifies the database you are using.

rule-numb

Specifies an integer between 1 and 255 that uniquely identifies this set of rules on your system that you created with the WBREAK-COMPILER qualifier.

NOTES

- If you change the word-break rules when word indexes are active, the indexes might not work properly because the rules used to create the index differ from those used when searching the index. Therefore, when you change the break rules for a database, you are warned if any word indexes are active. You should rebuild any such indexes. You can make old indexes consistent with the new rules by rebuilding them with the PROUTIL IDXBUILD qualifier. See Chapter 14, "Managing Performance," for more information about using the IDXBUILD qualifier.
- Progress maintains a CRC code to ensure that the word-break rule file does not change between sessions. If it has changed, Progress displays a message when you attempt to connect to the database. The connect attempt fails. You can fix this by restoring the original file or resetting the break rules to the default. Note that resetting to the default break rules might invalidate your word indexes.

RFUTIL Utility

Performs various roll-forward recovery activities, depending on the qualifier you supply.

SYNTAX

Operating System	Syntax
UNIX Windows	`rfutil` *db-name* `-C` *qualifier*

db-name

Specifies the database you are using.

`-C`

Specifies a particular utility or function when you use the `rfutil` command.

qualifier

Specifies the qualifier that you want to use. You can supply the qualifiers listed in Table 19–8.

NOTE: RFUTIL and its qualifiers support the use of internationalization startup parameters such as `-cpinternal` *codepage* and `-cpstream` *codepage*. See Chapter 18, "Database Startup Parameters," for a description of each database-related internationalization startup parameter.

Table 19–8: RFUTIL Utility Qualifiers *(1 of 2)*

Qualifier	Description
AIMAGE AIOFF	Disables after-imaging for a database during maintenance.
AIMAGE BEGIN	Enables after-imaging for a database.
AIMAGE END	Disables after-imaging for a database.

Table 19–8: RFUTIL Utility Qualifiers *(2 of 2)*

Qualifier	Description
AIMAGE EXTENT EMPTY	Marks an AI extent as empty and informs the Progress database manager that the indicated AI extent has been manually backed up and is now free for reuse.
AIMAGE EXTENT FULL	Displays the pathname of the oldest filled file.
AIMAGE EXTENT LIST	Displays information about extents.
AIMAGE NEW	Switches after-imaging to the next AI extent.
AIMAGE SCAN	Scans any after-image (AI) file and displays information from that file.
AIMAGE TRUNCATE	Truncates all of the variable-length after-image (AI) extents and optionally sets the AI block size with the After-image Block Size (`-aiblocksize`) parameter.
MARK BACKEDUP	Marks the database file, indicating that you have just completed a backup of the database.
ROLL FORWARD	Reconstructs a database by applying to that database all notes stored in the after-image (AI) file.
ROLL FORWARD RETRY	Restarts the roll-forward operation on the after-image extent that was in the process of rolling forward. The retry operation finds the transaction in process at the time of failure and resumes rolling forward.

The following RFUTIL entries describe each of these qualifiers.

RFUTIL AIMAGE AIOFF

Disables after-imaging for a database in multi-user mode. You do not have to shut down the database to use AIMAGE AIOFF. Use AIMAGE AIOFF to:

- Perform scheduled maintenance on a database.Disabling after-imaging during a table move or index rebuild saves time and disk space.
- Prevent database crashes caused by a pending lack of disk space. Instead of switching AI extents, you can disable after-imagine.

CAUTION: If you disable after-imaging and the database crashes, you cannot roll forward.

SYNTAX

Operating System	Syntax
UNIX Windows	`rfutil` *db-name* `-C aimage aioff`

db-name

Specifies the database you are using.

NOTE

- To enable after-imaging after disabling it with the RFUTIL qualifier AIMAGE AIOFF, use the RFUTIL qualifier AIMAGE BEGIN.

RFUTIL AIMAGE BEGIN Qualifier

Enables after-imaging for a database. The AIMAGE BEGIN qualifier:

- Creates the after-image (AI area) that will hold after-image notes
- Labels the database as having after-imaging enabled

SYNTAX

Operating System	Syntax
UNIX Windows	`rfutil` *`db-name`* `-C aimage begin`

db-name

Specifies the database you are using.

NOTE

- The AIMAGE BEGIN qualifier fails if:
  - It cannot truncate an existing AI area
  - You have not marked the database as backed up since the last time the database was modified

RFUTIL AIMAGE END Qualifier

Disables after-imaging for a database.

SYNTAX

Operating System	Syntax
UNIX Windows	`rfutil` *db-name* `-C aimage end`

db-name

Specifies the database you are using

NOTES

- The AIMAGE END qualifier fails if after-imaging is not enabled.
- When you use the RFUTIL AIMAGE END qualifier on a database with AI areas, RFUTIL marks all of the AI areas as empty. This means that you cannot access any information that you did not back up for roll-forward recovery.

RFUTIL AIMAGE EXTENT EMPTY Qualifier

Marks an AI extent as empty and informs the Progress database manager that the indicated AI extent has been manually backed up and is now free for reuse.

SYNTAX

Operating System	Syntax
UNIX Windows	rfutil *db-name* -C aimage extent empty [*extent-number* | *extent-path*]

db-name

Specifies the database you are using.

extent-number

Specifies the number of the extent you want to mark as empty.

extent-path

Specifies the pathname of the extent you want to mark as empty.

NOTES

- Use RFUTIL AIMAGE EXTENT LIST or RFUTIL AIMAGE EXTENT FULL to determine the *extent-number* or *extent-pathname*.
- If you do not specify either an extent number or an extent path, RFUTIL marks the oldest full extent as empty.

RFUTIL AIMAGE EXTENT FULL Qualifier

Displays the pathname of the oldest filled file. This is the next file to be backed up.

SYNTAX

Operating System	Syntax
UNIX Windows	`rfutil` *db-name* `-C aimage extent full`

db-name

Specifies the database you are using.

NOTE

- Use RFUTIL AIMAGE EXTENT FULL in a script similar to the following:

UNIX

```
last_full='_rfutil mydb -C aimage extent full'
tar -cvf /dev/ai_archive $last_full
rfutil mydb -C aimage extent full $last_full
```

RFUTIL AIMAGE EXTENT LIST Qualifier

Displays the following information:

- **Extent Number** — Number of each file.
- **Extent Type** — Type of each file. This is either fixed length or variable length.
- **Extent Path** — Pathname of each file.
- **Extent Size** — Size of each file in 1K blocks.
- **Space Used** — Number of blocks of space used in each file.
- **Extent Status** — Status of each file. This is either empty, full, or busy.
- **Start/Date Time** — Time each file began logging AI notes. This is not applicable to empty files.

SYNTAX

Operating System	Syntax
UNIX Windows	`rfutil` *db-name* `-C aimage extent list`

db-name

Specifies the database you are using.

NOTE

The status of a file might be full even though there is space left over in the file. This can happen after an online backup because file switch-over occurs at the time of the backup, whether or not the current file is full. Progress still marks the file as full because, like a full file, it must be archived and marked empty before Progress can reuse it.

RFUTIL AIMAGE NEW Qualifier

Changes the status of the current AI extent to full and changes the status of the next AI extent to busy. Use this qualifier only when after-imaging is enabled and you have just backed up the AI area.

SYNTAX

Operating System	Syntax
UNIX Windows	`rfutil` *db-name* `-C aimage new`

db-name

Specifies the database you are using.

NOTES

- The AIMAGE NEW qualifier fails if:
  - The next extent is not empty
  - The database has no AI extents
- You can use this qualifier whether the database is offline or online.

RFUTIL AIMAGE SCAN Qualifier

Scans any after-image (AI) file and displays information from that file.

SYNTAX

Operating System	Syntax
UNIX Windows	`rfutil` *db-name* `-C aimage scan [ verbose ]` `-a ai-name`

db-name

Specifies the database you are using.

`verbose`

Provides more information from the AI area, including the transaction number, the date and time the transaction began or ended, and the user ID of the user who initiated the transaction. You might want to try this on a small test AI area before running it on the after-image file associated with your database.

`-a` *ai-name*

Identifies the AI area of the specified database.

NOTES

- The AIMAGE SCAN qualifier fails if:
  - You omit the After-image area (`-a`) parameter
  - It cannot open the AI area
- The specified database does not have to be the database that corresponds to the AI area. You can use a dummy database to use this command with an AI area for an online database.

RFUTIL AIMAGE TRUNCATE Qualifier

Truncates all of the variable-length after-image (AI) extents and optionally sets the AI block size with the After-image Block Size (-aiblocksize) parameter.

SYNTAX

Operating System	Syntax
UNIX Windows	rfutil *db-name* -C aimage truncate -aiblocksize *size*

db-name

Specifies the database you are using.

-aiblocksize *size*

Specifies the size of the AI blocks in each buffer, in kilobytes. The valid values are 1, 2, 4, 8, and 16. The block size cannot be smaller than the database block size.

NOTES

- After executing this command to change the AI block size, you must perform a full backup of the database before you can re-enable after-imaging. If you change the BI block size or cluster size before backing up the database, the block size of the backup will overwrite the changed block size when the backup is restored.
- Increasing the AI block size allows larger AI reads and writes. This can reduce I/O rates on disks where the AI areas are located. If your operating system can benefit from larger writes, this option can improve performance. Larger AI block size might also improve performance for roll-forward recovery processing.
- When you execute this command, after-imaging and two-phase commit must be disabled, and the database must be offline; otherwise, Progress returns an error message.

- After you change the AI block size, Progress uses the new block size in all database operations.
- Use the PROSTRCT STATISTICS qualifier to display the block sizes for a database.
- Typically, if you change the AI block size, you should also change the before-image (BI) block and cluster size; otherwise, the increased AI performance will cause a BI bottleneck.
- See Chapter 14, "Managing Performance," for more information about using the RFUTIL AIMAGE TRUNCATE utility.

RFUTIL MARK BACKEDUP Qualifier

Marks the database file, indicating that you have just completed a backup of the database.

Use the MARK BACKEDUP qualifier if you are using an operating system backup utility instead of PROBKUP. The Progress Backup/Restore utility automatically marks the database as backed up.

SYNTAX

Operating System	Syntax
UNIX Windows	`rfutil` *`db-name`* `-C mark backedup`

db-name

Specifies the database you want to mark as backed up.

RFUTIL ROLL FORWARD Qualifier

Reconstructs a database by applying to that database all notes stored in the after-image (AI) file. The ROLL FORWARD qualifier displays the following information:

- The start and end dates of the AI area being applied to the database
- The number of completed transactions reapplied to the database
- The number of transactions that were active after all AI notes were applied.

SYNTAX

Operating System	Syntax
UNIX Windows	rfutil *db-name* -C roll forward [verbose] [endtime *yyyy:mm:dd:hh:mm:ss* | endtrans *transaction-number*] [-B *n*] [-r] -a *ai-name*

db-name

Specifies the database you are using.

verbose

Produces information for every note in the AI area.

-a *ai-name*

Identifies the AI area of the specified database.

endtime

Specifies to roll forward to a certain time. You must specify the ending time as a string of digits and separate the date and time components with a colon. Transactions are included in the partial roll forward only if they end before the specified time. For example, to roll forward to 5:10 PM on July 18, 2002, type **2002:07:18:17:10:00**. For Progress to include a transaction in this partial roll forward, the transaction must have ended on or before 2002:07:18:17:09:59.

`endtrans`

Specifies to roll forward up to but not including the transaction beginning that contains the *transaction-number*. For example, if you specify endtrans 1000, Progress rolls forward the AI area to transaction 999. If you want to include transaction 1000, you must specify endtrans 1001.

`-B` *n*

Specifies the number of database buffers. The single-user default value is 20.

`-r`

Directs Progress to use buffered I/O.

If the system crashes while you are running the ROLL FORWARD operation, restore your database files again and rerun the ROLL FORWARD operation.

The ROLL FORWARD qualifier fails if:

- You omit the After-image Filename (-a) parameter
- It cannot open the AI area
- You name the wrong AI area
- The database was opened before all AI extents were applied

NOTES

- The ROLL FORWARD qualifier always disables after-imaging for the database before beginning the roll-forward operation. After the roll-forward has completed, you must re-enable it with the AIMAGE BEGIN qualifier if you want continued AI protection.
- You must apply all AI extents associated with the database in the same sequence they were generated before you can use the database.

RFUTIL ROLL FORWARD RETRY Qualifier

Enhances the support of 24 X 7 database operations. The use of this qualifier is limited to RFUTIL roll-forward operations that fail because of power outages or system failures. The ROLL FORWARD RETRY qualifier restarts the roll-forward operation on the after-image extent that was in the process of rolling forward. The retry operation finds the transaction in process at the time of failure and resumes rolling forward. It recovers the (transaction log) TL extents. Entries noting the use of the retry operation appear in the LG file of the database. If subsequent failures occur during the retry operation, the retry operation can be restarted.

SYNTAX

Operating System	Syntax
UNIX Windows	rfutil *db-name* -C roll forward retry [endtime *yyyy:mm:dd:hh:mm:ss* | endtrans *transaction-number*] [-B *n*] [-r] [verbose] -a *ai-area*

db-name

Specifies the database you are using.

endtime

Specifies to roll forward to a certain point in time. You must specify the ending time as a string of digits and separate the date and time components with a colon. Transactions are included in the partial roll forward only if they end before the specified time. For example, to roll forward to 5:10 PM on July 18, 2002, type **2002:07:18:17:10:00**. For Progress to include a transaction in this partial roll forward, the transaction must have ended on or before 2002:07:18:17:09:59.

endtrans

Specifies to roll forward up to but not including the transaction beginning that contains the *transaction-number*. For example, if you specify endtrans 1000, Progress rolls forward the AI area to transaction 999. If you want to include transaction 1000, you must specify endtrans 1001.

`-B n`

Specifies the number of database buffers. The single-user default value is 20.

`-r`

Directs Progress to use buffered I/O.

verbose

Produces one line of information for every note in the AI area.

`-a ai-area`

Identifies the AI area of the specified database.

NOTE

- Roll forward might encounter a 2phase begin note in a BI area that will signal roll forward to enable transaction commit logging to the transaction log. If the database does not contain a transaction log (TL) area, roll forward will abort. To recover from this situation, you should first add a TL area to your database and then run ROLL FORWARD RETRY.

SQLDUMP Utility

A command-line utility that dumps application data from SQL-92 tables into one or more files. You can load the data from the files into another database with the SQLLOAD utility. The SQLDUMP utility does not dump data from Progress 4GL tables.

Before you can execute SQLDUMP against a database server, the server must be configured to accept SQL connections and must be running. See Chapter 4, "Creating and Deleting Databases," for instructions on creating a database and Chapter 5, "Starting Up and Shutting Down," for information about starting a Progress database or database server.

SYNTAX

Operating System	Syntax
UNIX Windows	`sqldump -u` *user_name* **[** `-a` *password* **]** **[** `-C` *code-page-name* **]** `-t` **[** *owner_name.***]** *table_name1* **[** **[**`,`*owner_name.***]** *table_name2*`,` **...** **]** *database_name*

`-u` *user_name*

Specifies the user id SQLDUMP used to connect to the database. If you omit the `user_name` and `password` parameter values, SQLDUMP prompts you for the values. If you omit `user_name` and supply a `password`, SQLDUMP uses the value defined in the USER environment variable as the user_name value.

`-a` *password*

Specifies the *password* used by the database for authentication.

`-C code-page-name`

A case-insensitive character string that specifies the name of the dump file's code page. If the `-C` parameter specifies a code page name that is not valid, Progress reports a run-time error. If the `-C` parameter does not appear at all, the code page name defaults to the client's internal code page, which is:

- The value of the client's SQL_CLIENT_CHARSET environment variable, if set
- Otherwise, the name of the code page of the client's locale.

For example, you might use the `-C` parameter to have a Windows client using the MS1250 code page produce a dump file using the ISO8859-2 code page (to read later on a UNIX machine, perhaps). Although you can accomplish this by setting the client's SQL_CLIENT_CHARSET environment variable, using the `-C` parameter might be easier.

`-t owner_name.table_name`

Specifies a list of one or more tables to dump to a file. This parameter is required. Pattern matching is supported in both *owner_name* and *table_name*, using a percent sign (%) for one or more characters and an underscore (_) for a single character. The pattern matching follows the standard defined by the LIKE predicate in SQL-92.

You can dump a single table, a set of tables, or all tables. If you omit the optional *owner_name* qualifier, SQLDUMP uses the name specified by the `-u` parameter.

database_name

Specifies the database where you are dumping tables. You can dump tables from one database each time you invoke SQLDUMP. There is no option flag preceding the *database_name*. This parameter is required and must be the last parameter specified. The database name is specified like a JDBC-style URL: *db_type*`:T:`*host:portnum:dbname*`.`

The SQLDUMP utility writes user data in row order into ASCII records with variable-length format. The column order in the files is identical to the column order in the tables. The utility writes both format and content header records to the dump file. You can dump multiple tables in a single execution by specifying multiple table names, separated by commas. Make sure there are no spaces before or after commas in the table list.

Data for one table always goes to a single dump file. Each dump file corresponds to one database table. For example, if you specify 200 tables in the SQLDUMP command, you will create 200 dump files. The SQLDUMP utility assigns the filenames that correspond to the *owner_name* and *table_name* in the database, with the file extension `.dsql`. If a dump file for a specified table already exists, it will be overwritten and replaced. Dump files are created in the current working directory.

The format of the records in a dump file is similar to the Progress 4GL `.d` file format:

- Converts all values to character representation
- Delimits CHARACTER values with double quotes
- Can contain any embedded characters except for NULL values, allowing commas, newlines, and other control characters
- Uses two sets of double quotes to escape embedded double quotes
- Delimits NUMERIC and other noncharacter datatypes using a space
- Processes TIMESTAMP data as if it were CHARACTER data
- Has a size limit of 2K for a single column value
- Has a maximum record length of 32K for dump file records

Any error is a fatal error, and SQLDUMP halts the dumping process so that data integrity will not be compromised. SQLDUMP reports errors to standard output.

After successful processing, SQLDUMP writes a summary report to standard output. For each table SQLDUMP processes, the report shows:

- Table name
- Dump filename
- Number of records dumped
- Number of bytes dumped
- Number of seconds required for processing

EXAMPLES

This example directs the SQLDUMP utility to write the data from two tables to two dump files. The *user_name* and *password* for connecting to the database are tucker and sulky. The tucker account must have the authority to access the customers and products tables in database salesdb with *owner_name* martin:

```
sqldump -u tucker -a sulky -t martin.customers,martin.products
progress:T:thunder:4077:salesdb
```

This example directs the SQLDUMP utility to write the data from all tables in the salesdb database that begin with any of these strings: cust, invent, and sales, and having any owner name that the user tucker has authority to access. The *user_name* and *password* for connecting to the database are tucker and sulky:

```
sqldump -u tucker -a sulky -t%.cust%,%.invent%,%.sales%
progress:T:thunder:4077:salesdb
```

This example directs the SQLDUMP utility to write the data from all tables for all owner names in the salesdb database:

```
sqldump -u tucker -a sulky -t %.%  progress:T:thunder:4077:salesdb
```

NOTES

- The *database_name* must be the last parameter given.
- Each dump file records character set information in the identifier section of each file. For example:

```
A^B^CProgress     sqlschema       v1.0       Quote fmt
A^B^CTimestamp    1999-10-19   19:06:49:0000
A^B^CDatabase     dumpdb.db
A^B^CProgress Character Set: iso8859-1
A^B^CJava Charcter Set: Unicode UTF-8
A^B^CDate Format: MM/DD/YYYY
```

The character set recorded in the dump file is the client character set. The default character set for all non-JDBC clients is taken from the local operating system through the operating system apis. JDBC clients use the Unicode UTF-8 character set.

To use a character set different than that used by the operating system, set the SQL_CLIENT_CHARSET environment variable to the name of the preferred character set. You can define any Progress supported character set name. The name is not case sensitive.

- SQLDUMP does not support the following characters in schema names:
  - Double quote (")
  - Forward slash (/)
  - Backslash (\)
- SQLDUMP, however, does support schema names that contain special characters such as, a blank space, a hyphen (-), or pound sign (#). These names must be used as delimited identifiers. Therefore, when specifying names with special characters on a UNIX command line, follow these three rules:
  - Use double quotes to delimit identifiers.
  - So that the command line does not strip the quotes, use a backslash (\) to escape the double quotes used for delimited identifiers.
  - Use double quotes to enclose any names with embedded spaces, commas, or characters special to a command shell (such as the Bourne shell). This use of quotes is in addition to quoting delimited identifiers.

  For example, to dump the table `Yearly Profits`, use the following UNIX command-line syntax:

```
sqldump -t "\"Yearly Profits\"" -u xxx -a yyy database_name
```

- On Windows, the command interpreter rules for the use of double quotation marks varies from UNIX.
- By default, SQLDUMP displays `promsgs` messages using the code page corresponding to *code-page-name*. That is, if you are dumping a Russian database, and *code-page-name* specifies the name of a Russian code page, the client displays `promsgs` messages using the Russian code-page, (unless you specify a different code page by setting the client's SQL_CLIENT_CHARSET_PROMSGS environment variable).

SQLLOAD Utility

A command-line utility that loads user data from a formatted file into an SQL-92 database. Typically, the source file for the load is created by executing the SQLDUMP utility. The SQLLOAD utility can process a source file created by another application or utility, if the format of the file conforms to SQLLOAD requirements. The file extension made available to SQLLOAD for processing must be .dsql. See the entry on SQLDUMP for a description of the required file format.

Before you can execute SQLLOAD against a database server, the server must be configured to accept SQL connections and must be running. See Chapter 4, "Creating and Deleting Databases," for instructions on creating a database and Chapter 5, "Starting Up and Shutting Down," for information about starting a Progress database or database server.

SYNTAX

Operating System	Syntax
UNIX Windows	sqlload -u *user_name* [-a *password*] -t [*owner_name.*]*table_name1* [[,*owner_name.*]*table_name2*, ...] [-l *log_file_name*] [-b *badfile_name*] [-e *max_errors*] [-s *skipcount*] [-m *maxrows*] [-F comma | quote] *database_name*

-u *user_name*

Specifies the user SQLLOAD uses to connect to the database. If you omit the *user_name* and *password*, SQLLOAD prompts you for these parameter values. If you omit the *user_name* and supply a password, SQLLOAD uses the value defined in the USER environment variable.

-a *password*

Specifies the *password* used by the database for authentication.

-C *code-page-name*

A case-insensitive character string that specifies the name of the dump file's code page. If the -C parameter specifies a code page name that is not valid, Progress reports a run-time error. If the -C parameter does not appear at all, the code page name defaults to the client's internal code page, which is:

- The value of the client's SQL_CLIENT_CHARSET environment variable, if set
- Otherwise, the name of the code page of the client's locale

For example, you might use the -C parameter to load a dump file whose code page is ISO8859-2, using a Windows client whose code page is MS1250. Although you can accomplish this by setting the client's SQL_CLIENT_CHARSET environment variable, using the -C parameter might be easier.

-t *owner_name.table_name*

Specifies a list of one or more tables to load into a database. This parameter is required. Pattern matching is supported, using a percent sign (%) for multiple characters and an underscore (_) for a single character. The pattern matching follows the standard for the LIKE predicate in SQL-92. You can load a single table, a set of tables, or all tables. If you omit the optional *owner_name* table qualifier, SQLLOAD uses the name specified by the -u parameter. The files from which SQLLOAD loads data are not specified in the SQLLOAD syntax. The utility requires that the filename follow the naming convention *owner_name.table_name*.dsql.

-l *log_file_name*

Specifies the file to which SQLLOAD writes errors and statistics. The default is standard output.

-b *badfile_name*

Specifies the file where SQLLOAD writes rows that were not loaded.

-e *max_errors*

Specifies the maximum number of errors that SQLLOAD allows before term processing. The default is 50.

-m *maxrows*

Directs SQLLOAD to stop loading rows at the specified number.

`-n`

Directs SQLLOAD to check for syntax errors without loading any rows.

`-F comma | quote`

Directs SQLLOAD to load data in comma-delimited format or quote-delimited format. The default is quote.

`database_name`

Identifies the database where you are loading tables. You can load tables into a single database each time you invoke SQLLOAD. There is no option flag preceding the *`database_name`*. This parameter is required, and must be the last parameter specified. The database name is specified like a JDBC-style URL: *`db_type:`*`T`*`:host:portnum:dbname`*.

The SQLLOAD utility reads application data from variable-length text-formatted files and writes the data into the specified database. The column order is identical to the table column order. SQLLOAD reads format and content header records from the dump file. You can load multiple tables in a single execution by specifying multiple table names, separated by commas. Data for one table is from a single dump file. Every source file corresponds to one database table. For example, if you specify 200 tables in the SQLLOAD command, you will load 200 database tables.

The format of the records in the input files is similar to the Progress 4GL `.d` file dump format. See the "SQLDUMP Utility" section in this chapter for a description of the record format. The maximum record length SQLLOAD can process is 32K.

Each database record read is share-locked, for consistency. You must ensure that the SQL Server has a lock table large enough to contain one lock for every record in the table. The default lock table size is 10,000 locks.

SQLLOAD writes any errors to standard output and halts the loading process for any error so that data integrity is not compromised.

EXAMPLES

This example directs the SQLLOAD utility to load the data from two dump files into the salesdb database. The input files to SQLLOAD must be tucker.customers.dsql and tucker.products.dsql:

```
sqlload -u tucker -a sulky -t tucker.customers,tucker.products
progress:T:thunder:4077:salesdb
```

This example directs SQLLOAD to load the data from all appropriately named dump files into the specified tables in the salesdb database:

```
sqlload -u tucker -a sulky -t %.cust%,%.invent%,%.sales%
progress:T:thunder:4077:salesdb
```

NOTES

- The *database_name* must be the last parameter given.
- The character set used by SQLLOAD must match the character set information recorded in each dump file. If the character sets do not match, the load is rejected. You can use the SQL_CLIENT_CHARSET environment variable to specify a character set.

  Each dump file you create with SQLDUMP contains character set information about that file. The character set recorded in the dump file is the client character set. The default character set for all non-JDBC clients is taken from the local operating system through the operating system apis. JDBC clients use the Unicode UTF-8 character set.

  To use a character set different than that used by the operating system, set the SQL_CLIENT_CHARSET environment variable to the name of the preferred character set. You can define any Progress-supported character set name. The name is not case sensitive.
- At run time, SQLLOAD reports an error if it detects a mismatch between the code page of the dump file being loaded and the code page of the client running SQLLOAD.
- By default, SQLLOAD displays `promsgs` messages using the code page corresponding to *code-page-name*. That is, if you are restoring a Russian database and *code-page-name* specifies the name of a Russian code page, the client displays `promsgs` messages using the Russian code-page (unless you specify a different code page by setting the client's SQL_CLIENT_CHARSET_PROMSGS environment variable).

- SQLLOAD does not support the following characters in schema names:
  - Double quote (")
  - Forward slash (/)
  - Backslash (\)
- SQLLOAD, however, does support schema names that contain special characters, such as a blank space, a hyphen (-), or pound sign (#). These names must be used as delimited identifiers. Therefore, when specifying names with special characters on a UNIX command line, follow these rules:
  - Use double quotes to delimit identifiers.
  - So that the command line does not strip the quotes, use a backslash (\) to escape the double quotes used for delimited identifiers.
  - Use double quotes to enclose any names with embedded spaces, commas, or characters special to a command shell (such as the Bourne shell). This use of quotes is in addition to quoting delimited identifiers.

  For example, to load the table `Yearly Profits` use the following UNIX command-line syntax:

```
sqlload -u xxx -a yyy -t "\"Yearly Profits\"" database_name
```

- On Windows NT, the command interpreter rules for the use of double quotation marks varies from UNIX.

SQLSCHEMA Utility

A command-line utility that writes SQL-92 database schema components to an output file selectively. You can capture table definitions including table constraints, views, stored procedures including related privileges, and triggers. At the command line you specify which components to dump. To load database schema information into a database, use the SQL Explorer tool. See the *Progress SQL-92 Guide and Reference* for information about SQL Explorer.

Before you can execute SQLSCHEMA against a database server, the server must be configured to accept SQL connections and must be running. See Chapter 4, "Creating and Deleting Databases," for instructions on creating a database and Chapter 5, "Starting Up and Shutting Down," for information about starting a Progress database or database server.

SYNTAX

Operating System	Syntax
UNIX Windows	sqlschema -u *user_name* [-a *password*] [-t [*owner_name.*] *table_name1* [,*owner_name.*] *table_name2*, . . .] [-p [*owner_name.*] *procedure_name*, . . .] [-T [*owner_name.*] *trigger_name*, . . .] [-g [*owner_name.*] *table_name*, . . .] [-s [*owner_name.*] *table_name*, . . .] [-o *output_file_name*] *database_name*

-u *user_name*

Specifies the user id that SQLSCHEMA employs to connect to the database. If you omit the *user_name* and *password*, SQLSCHEMA prompts you for these values. If you omit the *user_name* and supply a password, SQLSCHEMA uses the value defined by the USER environment variable.

-a *password*

Specifies the *password* used by the database for authentication.

`-t` *owner_name.table_name*

A list of one or more tables you want to capture definitions for. Pattern matching is supported, using a percent sign (%) for multiple characters and an underscore (_) for a single character. The pattern matching follows the standard for the LIKE predicate in SQL-92. You can write the definition for a single table, a set of tables, or all tables. If you omit the optional *owner_name* table qualifier, SQLSCHEMA uses the name specified by the `-u` parameter.

`-p` *owner_name.procedure_name*

A list of one or more procedures you want to capture definitions for. The SQLSCHEMA utility supports pattern matching for multiple and single characters. See the *owner_name.table_name* parameter for an explanation of pattern matching. You can capture the definitions for a single procedure, a set of procedures, or all procedures. If you omit the optional *owner_name* table qualifier, SQLSCHEMA uses the name specified by the `-u` parameter.

`-T` *owner_name.trigger_name*

A list of one or more triggers you want to capture definitions for. The SQLSCHEMA utility supports pattern matching for multiple and single characters. See the *owner_name.table_name* parameter for an explanation of pattern matching. You can capture the definition for a single trigger, a set of triggers, or all triggers. If you omit the optional *owner_name* table qualifier, SQLSCHEMA uses the name specified by the `-u` parameter.

`-g` *owner_name.table_name*

A list of one or more tables whose related privileges are captured as grant statements. You can write grant statements for both column and table privileges. The utility supports pattern matching for this parameter.

`-s` *owner_name.table_name*

Specifies a list of one or more tables whose related synonyms are captured as create synonym statements. The utility supports pattern matching for this parameter.

`-o` *output_file_name*`.dfsql`

Specifies the output file where SQLSCHEMA writes the definitions. When specified, the file extension name must be `.dfsql`. If output_file_name is omitted, SQLSCHEMA writes the definitions to the screen.

database_name

Identifies the database from which SQLSCHEMA captures component definitions. You can process a single database each time you invoke SQLSCHEMA. There is no option flag preceding the *database_name*. This parameter is required and must be the last parameter specified. The database name is specified in a connection string, such as *db-type:*T*:host:portnum:dbname*.

The SQLSCHEMA utility cannot write definitions for Progress 4GL tables. Table definitions include the database area name for the table, derived from a scan of the area and objects. When SQLSCHEMA writes a table definition, it does not automatically write associated triggers, synonyms, or privileges. These must be explicitly specified on the command line. Capturing database schema requires privileges to access the requested components.

EXAMPLES

This example directs the SQLSCHEMA utility to write table definitions and trigger information. The output goes to the screen since no *output_file_name* is specified. Since the user_name and password are not specified, SQLSCHEMA will prompt the user for these values:

```
sqlschema -t tucker.customers,tucker.products -T
tucker.customers,tucker.products progress:T:thunder:4077:salesdb
```

This example directs the SQLSCHEMA utility to write table definitions to an output file named salesdbschema.dfsql:

```
sqlschema -u tucker -a sulky -t %.cust%,%.invent%,%.sales% -o
salesdbschema.dfsql progress:T:thunder:4077:salesdb
```

NOTE

Each output file created by the SQLSCHEMA utility records character set information about the contents of the file. When you use SQLSCHEMA to dump schema information from a database the schema is written-out in Unicode UTF-8.

20

Virtual System Tables

Virtual system tables give 4GL and SQL-92 applications access to the same type of database information that you collect with the Progress monitor (PROMON) utility. Virtual system tables (VSTs) enable an application to examine the status of a database and monitor its performance. With the database broker running, 4GL and SQL-92 applications can call a VST and retrieve the specified information as run-time data.

This chapter contains the following sections:

- Update Access To Virtual System Tables
- Virtual System Table Summaries
- Progress Virtual System Table Schema Descriptions

20.1 Update Access To Virtual System Tables

Progress provides the empty, demo, and sports2000 databases with the virtual system table schemas already loaded. As new virtual system tables are made available, you can update the schemas in a database. To update the schemas, run the following PROUTIL command before you start the database server:

```
proutil db-name -C updatevst
```

db-name

Specifies the database you are using.

20.2 Virtual System Table Summaries

This section summarizes the virtual system tables that Progress provides. Table 20–1 briefly describes each table and refers you to a more detailed schema description for each table.

Table 20–1: Progress Virtual System Tables *(1 of 6)*

Virtual System Table	Description	Where To Find In this Chapter
After-image Log Activity File (_ActAILog)	Displays after-image log activity, such as the number of after-image writes, records and bytes written, busy buffer waits, and log force waits.	Table 20–2
Before-image Log Activity File (_ActBILog)	Displays before-image log activity, such as the number of before-image reads and writes, records and bytes written, number of records and bytes read and the number of busy and empty buffer and log force waits.	Table 20–3
Buffer Activity File (_ActBuffer)	Displays the activity of the database buffer cache, such as the number of logical reads and writes, OS reads and writes, checkpoints, deferred writes, LRU skips and writes, and APW enqueues.	Table 20–4
Index Activity File (_ActIndex)	Displays index activity, such as the number of entry finds, creates, and deletes, the number of locked entries removed, and the numbers of split and free blocks.	Table 20–5

Table 20–1: Progress Virtual System Tables *(2 of 6)*

Virtual System Table	Description	Where To Find In this Chapter
Input/Output Activity File (_ActIOFile)	Displays information about input/output activity, including the number of reads, writes, and extends for each file.	Table 20–6
Input/Output Type Activity File (_ActIOType)	Displays information about types of input/output activity, such as database reads and writes, before-image and after-image reads, total reads, before-image and after-image writes, committed transactions, and database up time.	Table 20–7
Lock Table Activity File (_ActLock)	Displays Lock Table activity, including the number of share, exclusive, upgrade, Rec Get, and redundant requests; the number of exclusive, Rec Get, share, and upgrade grants; the number of exclusive, Rec Get, share, and upgrade waits; the number of downgrades, transactions committed, cancelled requests, and database up time.	Table 20–8
Other Activity File (_ActOther)	Displays information about miscellaneous activity, including the number of commits, undo operations, semaphore waits, master block flushes, and database up time.	Table 20–9
Page Writer Activity File (_ActPWs)	Displays information about asynchronous page writer (APW) activity, including the number of APW queue and database writes, checkpoint and scan writes, total database writes; the number of buffers scanned and checkpointed; the number of checkpoints, checkpoint flushes, and marks; the number of scan cycles, committed transactions, and database up time.	Table 20–10

Table 20–1: Progress Virtual System Tables *(3 of 6)*

Virtual System Table	Description	Where To Find In this Chapter
Record Activity File (_ActRecord)	Displays record activity information, including the number of bytes created, deleted, read, and updated; the number of fragments created, deleted, read, and updated; the number of records created, deleted, read, and updated; the number of transactions committed; and database up time.	Table 20–11
Server Activity File (_ActServer)	Displays server activity information, including the number of bytes sent and received, the number of messages sent and received, the number of queries received, the number of records sent and received, the number of query time slice switches, the number of transactions committed, and database up time.	Table 20–12
Space Allocation Activity File (_ActSpace)	Displays space allocation information, including the number of database extends, the number of times a block was used from and returned to the free chain, the number of times space was allocated for a record (from the rm chain or from the free chain) the number of bytes allocated for record fragments, the number of rm blocks examined or removed, the number of blocks added to the front or back of the rm chain, the number of moved blocks, the number of locked chain entries, the number of transactions committed, and database up time.	Table 20–13
Summary Activity File (_ActSummary)	Displays general information about database activity, including the number of transactions committed and rolled back; the number of records read, updated, created, and deleted; the number of record locks and waits; the number of database reads and writes; before-image and after-image information; and buffer information.	Table 20–14

Table 20–1: Progress Virtual System Tables *(4 of 6)*

Virtual System Table	Description	Where To Find In this Chapter
Area Status File (_AreaStatus)	Displays a variety of data about that status of areas.	Table 20–15
Block File (_Block)	Displays information about a specific block.	Table 20–16
Buffer Status File (_BuffStatus)	Displays status of Progress buffers, such as the number of buffers that are in the buffer cache, that are currently in use, that are empty, or that are on the LRU chain, page writer queue, or checkpoint queue.	Table 20–17
Checkpoint File (_Checkpoint)	Displays information about each checkpoint, including the checkpoint number and beginning time, the time required to complete the checkpoint, and the number of modified blocks scheduled to be written. The file also describes APW-written blocks.	Table 20–18
Database Connection File (_Connect)	Displays information about two-phase commit, batch user, and device connections.	Table 20–19
Database Status File (_DbStatus)	Displays a wide variety of status data.	Table 20–20
Database File Status File (_Filelist)	Displays the file name, file size, and the size of every database file and extent.	Table 20–21
Index Statistics File (_IndexStat)	Displays statistics on the number of accesses to a specific range of indexes.	Table 20–22
Latch Statistics File (_Latch)	Displays statistics on latch and latch queue activity.	Table 20–23
License Management (_License)	Provides information about the number of users and connections.	Table 20–24
Lock Table Status File (_Lock)	Displays the status of the lock table, including the user number, the user name, lock type, RECID number, flags, and chain.	Table 20–25

Table 20–1: Progress Virtual System Tables *(5 of 6)*

Virtual System Table	Description	Where To Find In this Chapter
Lock Request File (_LockReq)	Displays information about lock requests, including user name and number, record locks and waits, schema locks and waits, and transaction locks and waits.	Table 20–26
Logging File (_Logging)	Includes before-image and after-image logging information and two-phase commit status.	Table 20–27
Master Block File (_MstrBlk)	Displays before-image and after-image information about the master block and other master block status.	Table 20–28
User Connection (_MyConnection)	Provides information about a user and the number of private read-only buffers allowed and in use.	Table 20–29
Resource Queue Statistics File (_Resrc)	Displays statistics on resource queue utilization.	Table 20–30
Segments File (_Segments)	Reports the segment number, segment size, and the number of free and used segments.	Table 20–31
Servers File (_Servers)	Displays status of Progress servers running on the system, such as the server number, process ID, and type; the protocol used; the number of logins and current users; the maximum number of users; and the server's port number.	Table 20–32
Startup File (_Startup)	Displays the values of the database startup parameters that were used.	Table 20–33
Index and Table Statistics Range (_StatBase)	Displays basic table and index statistics.	Table 20–34
Table Statistics File (_TableStat)	Displays statistics on the number of accesses to a specific range of tables.	Table 20–35

Table 20–1: Progress Virtual System Tables *(6 of 6)*

Virtual System Table	Description	Where To Find In this Chapter
Transaction File (_Trans)	Includes information such as transaction number, state, start time, duration, user number, coordinator name, and transaction.	Table 20–36
Transaction End Lock Statistics (_TxeLock)	Includes statistics about Transaction End Locks.	Table 20–37
Database Input/Output File (_UserIO)	Displays information about the database input/output operations, including user number and name and the number of accesses, reads, and writes.	Table 20–38
Record Locking Table File (_UserLock)	Displays the contents of the record locking table, such as user name, chain, number, record ID, lock type, and flags.	Table 20–39
User Status (_UserStatus)	Displays the progress of the IDXMOVE, TABLEMOVE, and INDEXCOMPACT utilities, such as the phase the utility is in, the user number, the tables and indexes worked on, the current position in the record chain, and the number of blocks accessed, records moved, and tables and indexes compacted.	Table 20–40

20.3 Progress Virtual System Table Schema Descriptions

Table 20–2 through Table 20–40 describe the schema for each virtual system table.

Table 20–2: After-image Log Activity File (_ActAILog)

Record	Description
_AiLog-AIWWrites	Number of after-image (AI) writes performed by the after-image writer (AIW). This is a subset of the total number of AI writes.
_AiLog-BBuffWaits	Number of busy buffer waits.
_AiLog-BytesWritn	Amount of AI data written to the AI file, in bytes.
_AiLog-ForceWaits	Number of waiting-for-commit records to be written to disk.
_AiLog-NoBufAvail	Total number of times a process had to wait because a buffer was not available.
_AiLog-PartialWrt	Number of writes to the AI file made before the AI buffer is full.
_AiLog-RecWriten	Number of records written to the AI file.
_AiLog-TotWrites	Total number of writes to the AI file.
_AiLog-Trans	Number of transactions committed to the AI file.
_AiLog-UpTime	Number of seconds the AI file was open.

Table 20–3: Before-image Log Activity File (_ActBILog) *(1 of 2)*

Record	Description
_BiLog-BBuffWaits	Number of times a process had to wait for a buffer that was busy.
_BiLog-BIWWrites	Number of writes to the BI file performed by the before-image writer (BIW). For good performance, this number should be high in relation to the total number of BI writes.
_BiLog-BytesRead	Number of bytes of data read from the BI file.
_BiLog-BytesWrtn	Number of bytes of data written to the BI file.
_BiLog-ClstrClose	Number of BI clusters filled and closed in preparation for reuse.
_BiLog-EBuffWaits	Number of times a process had to wait because all buffers were full.
_BiLog-ForceWaits	Number of waiting-for-commit records to be written to disk.
_BiLog-ForceWrts	Number of waiting-for-commit records written to disk.
_BiLog-PartialWrts	Number of writes to the BI file made before the BI buffer is full.
_BiLog-RecRead	Number of BI records (notes) read from the BI file.
_BiLog-RecWriten	Number of BI records (notes) written to the BI file.
_BiLog-TotalWrts	Number of total writes to the BI file.
_BiLog-TotReads	Number of BI blocks read from the BI file to undo transactions.

Table 20–3: Before-image Log Activity File (_ActBILog) *(2 of 2)*

Record	Description
_BiLog-Trans	Number of transactions committed to the BI file.
_BiLog-UpTime	Number of seconds the BI file was open.

Table 20–4: Buffer Activity File (_ActBuffer) *(1 of 2)*

Record	Description
_Buffer-APWEnq	Number of modified buffers placed on the APW queue for writing.
_Buffer-Chkpts	Number of checkpoint operations.
_Buffer-Deferred	Total number of changes to blocks that occurred before the blocks were written. Each deferred write is potentially an I/O operation saved.
_Buffer-Flushed	Number of blocks that were not written during the checkpoint and that had to be written all at once at the end of the checkpoint.
_Buffer-LogicRds	Number of client requests for database block read operations.
_Buffer-LogicWrts	Number of client requests for database block write operations.
_Buffer-LRUSkips	Number of times a buffer on the LRU chain was skipped because it was locked or modified.
_Buffer-LRUwrts	Number of blocks written to free a buffer for a read operation.
_Buffer-Marked	Number of blocks scheduled to be written before the end of a checkpoint.
_Buffer-OSRds	Number of database blocks read from disk.

Table 20–4: Buffer Activity File (_ActBuffer) *(2 of 2)*

Record	Description
_Buffer-OSWrts	Number of database block writes to disk.
_Buffer-Trans	Number of transactions committed.
_Buffer-Uptime	Number of seconds the database was opened.

Table 20–5: Index Activity File (_ActIndex)

Record	Description
_Index-Create	Number of new index entries generated.
_Index-Delete	Number of index entries deleted.
_Index-Find	Number of times an index entry was looked up.
_Index-Free	Number of index blocks freed during deletes.
_Index-Remove	Number of locks released at transaction end.
_Index-Splits	Number of block splits resulting from index additions.
_Index-Trans	Number of transactions committed.
_Index-UpTime	Number of seconds the database was up.

Table 20–6: Input/Output Activity File (_ActIOFile) *(1 of 2)*

Record	Description
_IOFile-BufReads	Number of buffered read operations performed on the file.
_IOFile-BufWrites	Number of buffered write operations performed on the file.

Table 20–6: Input/Output Activity File (_ActIOFile) *(2 of 2)*

Record	Description
_IOFile-Extends	Number of extend operations performed on the file.
_IOFile-FileName	Name of the file upon which operations are performed.
_IOFile-Reads	Number of read operations performed on the file.
_IOFile-Trans	Number of transactions committed.
_IOFile-UbufReads	Number of unbuffered read operations performed on the file.
_IOFile-UbufWrites	Number of unbuffered write operations performed on the file.
_IOFile-UpTime	Number of seconds the database was up.
_IOFile-Writes	Number of write operations performed on the file.

Table 20–7: Input/Output Type Activity File (_ActIOType) *(1 of 2)*

Record	Description
_IOType-AiRds	Number of AI read operations.
_IOType-AiWrts	Number of AI write operations.
_IOType-BiRds	Number of BI read operations.
_IOType-BiWrts	Number of BI write operations.
_IOType-DataReads	Number of reads of data blocks.
_IOType-DataWrts	Number of writes to data blocks.
_IOType-IdxRds	Number of reads of index blocks.
_IOType-IdxWrts	Number of writes to index blocks.

Table 20–7: Input/Output Type Activity File (_ActIOType) *(2 of 2)*

Record	Description
_IOType-Trans	Number of transactions committed.
_IOType-UpTime	Number of seconds the database was up.

Table 20–8: Lock Table Activity File (_ActLock) *(1 of 2)*

Record	Description
_Lock-CanclReq	Number of lock requests that were cancelled.
_Lock-Downgrade	Number of locks that were downgraded.
_Lock-ExclFind	Number of exclusive find requests granted.
_Lock-ExclLock	Number of exclusive lock requests.
_Lock-ExclReq	Number of requests for exclusive locks granted.
_Lock-ExclWait	Number of times processes waited for an exclusive lock.
_Lock-RecGetLock	Number of requests for record get locks granted.
_Lock-RecGetReq	Number of record get locks granted.
_Lock-RecGetWait	Number of times processes waited for record gets.
_Lock-RedReq	Number of duplicate requests granted.
_Lock-ShrFind	Number of share find requests granted.
_Lock-ShrLock	Number of share lock requests granted.
_Lock-ShrReq	Number of requests for share locks.
_Lock-ShrWait	Number of times processes waited for share lock.

Table 20–8: Lock Table Activity File (_ActLock) *(2 of 2)*

Record	Description
_Lock-Trans	Number of transactions committed.
_Lock-UpgLock	Number of requests for lock upgrades granted.
_Lock-UpgReq	Number of requests to upgrade a lock from shared lock to exclusive lock.
_Lock-UpgWait	Number of times processes waited for upgrades.
_Lock-UpTime	Number of seconds the database was up.

Table 20–9: Other Activity File (_ActOther)

Record	Description
_Other-Commit	Number of transactions committed.
_Other-FlushMblk	Number of times the database master block was written to disk.
_Other-Trans	Transactions committed.
_Other-Undo	Number of transactions rolled back.
_Other-UpTime	Number of seconds the database was up.
_Other-Wait	Number of times a process had to wait for a resource.

Table 20–10: Page Writer Activity File (_ActPWs)

Record	Description
_PW-ApwQWrites	Number of buffers written to clear the APW queue.
_PW-BuffsScaned	Number of buffers scanned during each cycle.
_PW-BufsCkp	Number of buffers checkpointed.
_PW-Checkpoints	Number of checkpoints that have occurred.
_PW-CkpQWrites	Number of buffers written from the checkpoint queue.
_PW-DBWrites	Number of database write operations performed by the APW.
_PW-Flushed	Number of blocks that were not written during the checkpoint and had to be written all at once at the end of the checkpoint.
_PW-Marked	Number of buffers that were scheduled to be written before the end of the checkpoint.
_PW-ScanCycles	Number of scan cycles. During a scan cycle, the APW scans a portion of the buffer pool to look for modified buffers.
_PW-ScanWrites	Number of buffers written during the scan cycle.
_PW-TotDBWrites	Total number of database write operations performed by all processes.
_PW-Trans	Number of transactions committed.
_PW-UpTime	Number of seconds the database was up.

Table 20–11: Record Activity File (_ActRecord)

Record	Description
_Record-BytesCreat	Number of bytes created.
_Record-BytesDel	Number of bytes deleted.
_Record-BytesRead	Number of bytes read.
_Record-BytesUpd	Number of bytes updated.
_Record-FragCreat	Number of fragments created.
_Record-FragDel	Number of fragments deleted.
_Record-FragRead	Number of fragments read.
_Record-FragUpd	Number of fragments updated.
_Record-RecCreat	Number of records created.
_Record-RecDel	Number of records deleted.
_Record-RecRead	Number of records read.
_Record-RecUpd	Number of records updated.
_Record-Trans	Number of transactions committed.
_Record-UpTime	Number of seconds the database was up.

Table 20–12: Server Activity File (_ActServer) *(1 of 2)*

Record	Description
_Server-ByteRec	Number of bytes received by the server.
_Server-ByteSent	Number of bytes sent by the server.
_Server-MsgRec	Number of network packets received by the server.

Table 20–12: Server Activity File (_ActServer) *(2 of 2)*

Record	Description
_Server-MsgSent	Number of network packets sent by the server.
_Server-QryRec	Number of query requests received.
_Server-RecRec	Number of records received.
_Server-RecSent	Number of records sent.
_Server-TimeSlice	Number of query time slice switches.
_Server-Trans	Number of transactions committed.
_Server-UpTime	Number of seconds the database was up.

Table 20–13: Space Allocation Activity File (_ActSpace) *(1 of 2)*

Record	Description
_Space-AllocNewRm	Number of times space was allocated for a record or record fragment.
_Space-BackAdd	Number of blocks added to the back of the rm chain.
_Space-BytesAlloc	Number of bytes allocated for record fragments.
_Space-DbExd	Number of times the database was extended.
_Space-Examined	Number of blocks examined in the rm chain while looking for space for a record fragment.
_Space-FromFree	Number of times space was allocated from the free chain.
_Space-FromRm	Number of times space was allocated from the rm chain.

Table 20–13: Space Allocation Activity File (_ActSpace) *(2 of 2)*

Record	Description
_Space-Front2Back	Number of blocks moved from the front to the back of the rm chain.
_Space-FrontAdd	Number of blocks added to the front of the rm chain.
_Space-Locked	Number of locked rm entries removed.
_Space-Removed	Number of blocks removed from the rm chain.
_Space-RetFree	Number of times a block was returned to the free chain.
_Space-TakeFree	Number of times a block was used from the free chain.
_Space-Trans	Number of transactions committed.
_Space-UpTime	Number of seconds the database was up.

Table 20–14: Summary Activity File (_ActSummary) *(1 of 2)*

Record	Description
_Summary-AiWrites	Number of AI blocks written to disk.
_Summary-BiReads	Number of BI blocks read.
_Summary-BiWrites	Number of BI blocks written to disk.
_Summary-Chkpts	Number of checkpoints that have been performed.
_Summary-Commits	Number of transactions all users have committed.
_Summary-DbAccesses	Number of times users have waited for shared and exclusive locks on a database buffer.

Table 20–14: Summary Activity File (_ActSummary) *(2 of 2)*

Record	Description
_Summary-DbReads	Number of database blocks read.
_Summary-DbWrites	Number of database blocks written to disk.
_Summary-Flushed	Number of database buffers that have been flushed to disk because they were not written by the time the checkpoint ended.
_Summary-RecCreat	Number of records created.
_Summary-RecDel	Number of records deleted.
_Summary-RecLock	Number of record locks used.
_Summary-RecReads	Number of records read.
_Summary-RecUpd	Number of records updated.
_Summary-RecWait	Number of times users have waited to access a locked record.
_Summary-TransComm	Number of transactions committed.
_Summary-Undos	Number of transactions rolled back.
_Summary-Uptime	Number of seconds the database was up.

Table 20–15: Area Status File (_AreaStatus) *(1 of 2)*

Record	Description
_AreaStatus-Areanum	Area number.
_AreaStatus-Areaname	Area name.
_AreaStatus-Totblocks	Total number of blocks in an area.
_AreaStatus-Highwater	Highest block used.
_AreaStatus-Extents	Number of extents in an area.

Table 20–15: Area Status File (_AreaStatus) *(2 of 2)*

Record	Description
_AreaStatus-Lastextent	Name of extent where high-water mark resides.
_AreaStatus-Freenum	Number of blocks on the free chain.
_AreaStatus-Rmnum	Number of blocks on the rm chain.

Table 20–16: Block File (_Block)

Record	Description
_Block-Area	Block area number.
_Block-BkupCtr	Backup counter.
_Block-Block	Data section of the block.
_Block-ChainType	Chain type.
_Block-Dbkey	Dbkey.
_Block-NextDbkey	Next Dbkey in the appropriate chain.
_Block-Type	Type of block.
_Block-Update	Number of times the block has been updated.

Table 20–17: Buffer Status File (_BuffStatus) *(1 of 2)*

Record	Description
_BfStatus-APWQ	Number of buffers on the page writer queue.
_BfStatus-CKPMarked	Number of buffers currently marked for checkpoint.
_BfStatus-CKPQ	Number of buffers on the checkpoint queue.
_BfStatus-HashSize	Size of the buffer hash table.

Table 20–17: Buffer Status File (_BuffStatus) *(2 of 2)*

Record	Description
_BfStatus-LastCkpNum	Most recent checkpoint number. As checkpoints begin, they are assigned a sequential number from the start of the session. The number is also the number of checkpoints that have occurred.
_BfStatus-LRU	Number of buffers on the least recently used (LRU) chain.
_BfStatus-ModBuffs	Number of dirty (modified) buffers.
_BfStatus-TotBufs	Number of buffers in the buffer cache.
_BfStatus-UsedBuffs	Number of buffers used.

Table 20–18: Checkpoint File (_Checkpoint)

Record	Description
_Checkpoint-ApwQ	Number of blocks written by the APW queue and replaced on the least recently used (LRU) chain by APWs.
_Checkpoint-CptQ	Number of blocks written from the checkpoint queue by the APWs.
_Checkpoint-Dirty	Number of modified blocks scheduled to be written.
_Checkpoint-Flush	Total number of blocks not written during the checkpoint that had to be written all at once at the end of the checkpoint.
_Checkpoint-Len	Length of time required to complete the checkpoint.
_Checkpoint-Scan	Number of blocks written by the APWs during the scan cycle.
_Checkpoint-Time	Time the checkpoint began.

Table 20–19: Database Connection File (_Connect)

Record	Description
_Connect-2phase	Displays a two-phase commit flag, either “yes” or “no.”
_Connect-Batch	Displays batch users status.
_Connect-Device	Device from which the user has connected to the database.
_Connect-Disconnect	Displays a disconnect flag.
_Connect-Interrupt	Displays an interrupt flag.
_Connect-Name	Displays the user name used for the connection.
_Connect-Pid	Displays the process ID of the user session.
_Connect-Resync	Displays a flag that indicates the database is resyncing.
_Connect-SemId	Displays the semaphore ID.
_Connect-SemNum	Displays the semaphore number.
_Connect-Server	Identifies the server if the connection is remote.
_Connect-Time	Time the user connected to the database.
_Connect-transId	Displays the current transaction ID.
_Connect-Type	Displays the connection type: SELF, REMC, BROK, SERV, or BTCH.
_Connect-Usr	Displays the user number.
_Connect-Wait	Displays the shared memory resource the user is waiting on.
_Connect-Wait1	Displays additional wait flags.

Table 20–20: Database Status File (_DbStatus) *(1 of 2)*

Record	Description
_DbStatus-AiBlkSize	Number of bytes in an after-image block.
_DbStatus-BiBlkSize	Number of bytes in a before-image block.
_DbStatus-BiClSize	Number of kilobytes in the before-image cluster.
_DbStatus-BiOpen	Most recent before-image (.bi) file open.
_DbStatus-BiSize	Logical size of the BI file. You can use this in conjunction with the _DbStatus-BiClSize value to determine the number of clusters in the BI file.
_DbStatus-BiTrunc	Time since the last truncate.
_DbStatus-CacheStamp	Schema cache file time stamp.
_DbStatus-Changed	Flag indicating if database has changed since last backup.
_DbStatus-ClVersMinor	Client minor version number.
_DbStatus-Codepage	Database character set.
_DbStatus-Collation	Database collation name.
_DbStatus-CreateDate	Date and time of database creation.
_DbStatus-DbBlkSize	Database block size.
_DbStatus-DbVers	Database version number.
_DbStatus-DbVersMinor	Database minor version number.
_DbStatus-EmptyBlks	Number of empty blocks in the database.
_DbStatus-FbDate	Most recent full backup.
_DbStatus-FreeBlks	Number of free blocks in the database.
_DbStatus-HiWater	Database blocks high-water mark.

Table 20–20: Database Status File (_DbStatus) *(2 of 2)*

Record	Description
_DbStatus-IbDate	Most recent incremental backup.
_DbStatus-IbSeq	Sequence of last incremental backup.
_DbStatus-Integrity	DbStatus-Integrity enabled flag (-i).
_DbStatus-IntFlags	Integrity flags.
_DbStatus-LastOpen	Most recent database open.
_DbStatus-LastTable	Highest table number defined.
_DbStatus-LastTran	Last transaction ID.
_DbStatus-MostLocks	Lock table high-water mark.
_DbStatus-NumAreas	Number of areas.
_DbStatus-NumLocks	Lock table entries in use.
_DbStatus-NumSems	Number of semaphores.
_DbStatus-PrevOpen	Previous database open.
_DbStatus-RMFreeBlks	Number of rm blocks with free space.
_DbStatus-SharedMemVer	Shared memory version number.
_DbStatus-ShmVers	Shared memory version number.
_DbStatus-Starttime	Time when the database was started.
_DbStatus-State	State of the database.
_DbStatus-Tainted	Damaged database flags.
_DbStatus-TotalBlks	Number of blocks allocated to the database.

Table 20–21: Database File Status File (_Filelist)

Record	Description
_FileList-BlkSize	Block size of the file.
_FileList-Extend	Amount of most recent extend in blocks.
_FileList-LogicalSz	Logical file size, in blocks.
_FileList-Name	Name of the file.
_FileList-Openmode	Displays the mode in which the file is opened.
_FileList-Size	Size of the file.

Table 20–22: Index Statistics File (_IndexStat)

Record	Description
_IndexStat-Blockdelete	Number of block deletes that have occurred to the index.
_IndexStat-Delete	Number of times delete access has occurred to the index.
_IndexStat-Create	Number of times create access has occurred to the index.
_IndexStat-Read	Number of times read access has occurred to the index.
_IndexStat-Split	Number of split operations that have occurred to the index.

Table 20–23: Latch Statistics File (_Latch)

Record	Description
_Latch-Busy	Latch busy Count.
_Latch-Hold	User number of the last latch holder.
_Latch-Lock	Latch lock count.
_Latch-LockedT	Used for calculation of duration of lock.
_Latch-LockT	Number, in microseconds, the latch was locked.
_Latch-Name	Latch name.
_Latch-Qhold	User number of the last queue holder.
_Latch-Spin	Latch spin count.
_Latch-Type	Latch type.
_Latch-Wait	Latch wait (nap) count.
_Latch-WaitT	Number, in microseconds, of wait or spin time.

Table 20–24: License Management (_License) *(1 of 2)*

Record	Description
_Lic-ActiveConns	Number of active connections.
_Lic-BatchConns	Number of current batch connections.
_Lic-CurrConns	Number of current connections.
_Lic-MaxActive	Maximum number of active connections.
_Lic-MaxBatch	Maximum number of batch connections.
_Lic-MaxCurrent	Maximum number of current connections.

Table 20–24: License Management (_License) *(2 of 2)*

Record	Description
_Lic-MinActive	Minimum number of active users.
_Lic-MinBatch	Minimum number of batch connections.
_Lic-MinCurrent	Minimum number of current connections.
_Lic-ValidUsers	Number of valid license users.

Table 20–25: Lock Table Status File (_Lock)

Record	Description
_Lock-Chain	Chain number.
_Lock-Flags	Flags for the lock. The flags specify a share lock (S), exclusive lock (X), a lock upgraded from share to exclusive (U), a lock in limbo (L), or a queued lock (Q).
_Lock-Name	User name.
_Lock-RecId	RECID of the record and lock table entry.
_Lock-Table	Table name.
_Lock-Type	Type of lock (for example, REC, RGET).
_Lock-Usr	User number of the user who owns the lock entry.

Table 20–26: Lock Request File (_LockReq) *(1 of 2)*

Record	Description
_LockReq-ExclFind	Number of exclusive lock finds acquired by the user.
_LockReq-Name	User name.

Table 20–26: Lock Request File (_LockReq) *(2 of 2)*

Record	Description
_LockReq-Num	User number.
_LockReq-RecLock	Number of record locks acquired by the user.
_LockReq-RecWait	Number of times the user had to wait for a record lock.
_LockReq-SchLock	Number of schema locks acquired by the user.
_LockReq-SchWait	Number of times the user had to wait for a schema lock.
_LockReq-ShrFind	Number of share lock finds acquired by the user.
_LockReq-TrnLock	Number of transaction locks acquired by the user.
_LockReq-TrnWait	Number of times the user had to wait for a transaction lock.

Table 20–27: Logging File (_Logging) *(1 of 3)*

Record	Description
_Logging-2PC	"Yes" indicates that two-phase commit is enabled; "No" indicates that two-phase commit is disabled.
_Logging-2PCNickName	Nickname that Progress uses to identify the coordinator database. You specify a nickname when you enable two-phase commit.
_Logging-2PCPriority	Priority for the coordinator database. You specify the priority when you enable two-phase commit.
_Logging-AiBegin	Date of the last AIMAGE BEGIN command.

Table 20–27: Logging File (_Logging) *(2 of 3)*

Record	Description
_Logging-AiBlkSize	Size of the after-image block. You can change the AI block size. This reduces I/O rates on disks that have AI files.
_Logging-AiBuffs	Number of buffers in the AI buffer pool. You can change the number of AI buffers with the After-image Buffers (`-aibufs`) startup parameter. The default value is 1.
_Logging-AiCurrExt	Current after-image extent.
_Logging-AiExtents	Number of AI files or extents.
_Logging-AiGenNum	Generation number of the current AI extent.
_Logging-AiIO	After-image I/O. “Reliable” indicates that synchronous or direct I/O is being used for AI writes. “Buffered” indicates that buffered I/O is being used for AI writes.
_Logging-AiJournal	“YES” indicates that after-imaging is enabled; “NO” indicates that after-imaging is disabled. Progress Software recommends that you use after-imaging if you use two-phase commit.
_Logging-AiLogSize	Number of kilobytes in the AI log file, if the AI extent in use is variable length. If the AI extent in use is fixed length, the value is 0.
_Logging-AiNew	Date of the last AIMAGE NEW command.
_Logging-AiOpen	Date when the AI log was last opened.
_Logging-BiBlkSize	BI block size.
_Logging-BiBuffs	Number of BI buffers.
_Logging-BiBytesFree	Number of free bytes remaining in the current BI cluster.

Table 20–27: Logging File (_Logging) *(3 of 3)*

Record	Description
_Logging-BiClAge	Period of time that must pass before Progress reuses a BI cluster. This period ensures that database blocks flushed at checkpoint are moved from the UNIX buffers on disk. When this occurs, the transaction is durably recorded on disk.
_Logging-BiClSize	BI cluster size.
_Logging-BiExtents	Number of BI extents.
_Logging-BiFullBuffs	Number of full BI buffers.
_Logging-BiIO	One of the following: “Reliable” indicates that synchronous or direct I/O is being used for BI writes; “BUFFERED” indicates that buffered I/O is being used for BI writes. Progress Software does not recommend buffered I/O.
_Logging-BiLogSize	Number of kilobytes in the BI file.
_Logging-CommitDelay	Current value of the Delayed BI File Write (`-Mf`) parameter.
_Logging-CrashProt	“Yes” indicates that crash protection is enabled; “No” indicates that crash protection is disabled.
_Logging-LastCkp	Time of the last checkpoint.

Table 20–28: Master Block File (_MstrBlk) *(1 of 2)*

Record	Description
_MstrBlk-AiBlksize	Number of bytes in an AI block.
_MstrBlk-BiBlksize	Number of bytes in a BI block.
_MstrBlk-BiOpen	Date and time the BI file was last opened.

Table 20–28: Master Block File (_MstrBlk) *(2 of 2)*

Record	Description
_MstrBlk-BiPrev	Previous value of _MstrBlk-biopen.
_MstrBlk-BiState	Current state of Bi file.
_MstrBlk-Cfilnum	Highest file number currently defined in the database.
_MstrBlk-Crdate	Date and time of database creation.
_MstrBlk-Dbstate	Current state of the database.
_MstrBlk-Dbvers	Database version number.
_MstrBlk-Fbdate	Date and time of the last full backup.
_MstrBlk-Hiwater	Database blocks highwater mark.
_MstrBlk-Ibdate	Date and time of most recent incremental backup.
_MstrBlk-Ibseq	Sequence of last incremental backup.
_MstrBlk-Integrity	Database integrity enabled flag.
_MstrBlk-Lasttask	Last transaction ID.
_MstrBlk-Oppdate	Date and time of previous database open.
_MstrBlk-Oprdate	Date and time of most recent database open.
_MstrBlk-Rlclsize	Current BI cluster size.
_MstrBlk-Rltime	Time since last before image file truncate.
_MstrBlk-Tainted	Database damaged flags.
_MstrBlk-Timestamp	Schema cache file time stamp.
_MstrBlk-Totblks	Number of blocks allocated to the database.

Table 20–29: User Connection (_MyConnection)

Record	Description
_MyConn-NumSeqBuffers	Number of private read-only buffers currently allowed.
_MyConn-Pid	This user's process ID.
_MyConn-UsedSeqBuffers	Number of private read-only buffers currently in use.
_MyConn-UserId	The user's user ID.

Table 20–30: Resource Queue Statistics File (_Resrc)

Record	Description
_Resrc-Name	Resource queue name.
_Resrc-Lock	Resource queue lock count.
_Resrc-Wait	Resource queue wait count.
_Resrc-Time	Resource queue wait time.

Table 20–31: Segments File (_Segments)

Record	Description
_Segment-ByteFree	Number of free bytes in this shared memory segment.
_Segment-BytesUsed	Number of bytes used in this shared memory segment.
_Segments-Segld	Segment ID.
_Segments-SegSize	Segment size.

Table 20–32: Servers File (_Servers)

Record	Description
_Server-CurrUsers	Number of current users connected to this server.
_Server-Logins	Number of established logins.
_Server-MaxUsers	Maximum number of logins available on the server.
_Server-Num	Server number.
_Server-Pid	Process ID of the server.
_Server-PortNum	TCP/IP port number of the server.
_Server-Protocol	Protocol supported by the server.
_Server-Type	Server type.

Table 20–33: Startup File (_Startup) *(1 of 2)*

Record	Description
_Startup-AiBuffs	Number of after-image buffers.
_Startup-AiName	Value of the After-image Filename (`-a`) startup parameter.
_Startup-APWBuffs	Value of the Page Writer Scan (`-pwscan`) parameter.
_Startup-APWMaxWrites	Value of the Page Writer Maximum Buffers (`-pwwmax`) startup parameter.
_Startup-APWQTime	Value of the Page Writer Queue Delay (`-pwqdelay`) startup parameter.
_Startup-APWSTime	Value of the Page Writer Scan Delay (`-pwsdelay`) parameter.
_Startup-BiBuffs	Number of before-image buffers.

Table 20–33: Startup File (_Startup) *(2 of 2)*

Record	Description
_Startup-BiDelay	Delay of before-image flush (-mf).
_Startup-BiIO	Before-image file I/O (-r, -R).
_Startup-BiName	Before-image filename (-g).
_Startup-BiTrunc	Before-image truncate interval (-G).
_Startup-Buffs	Number of database buffers (-B).
_Startup-CrashProt	No crash protection (-i).
_Startup-Directio	Direct I/O data transfer (-directio). Field is set to 1 when -directio is specified.
_Startup-LockTable	Maximum size of the locking table.
_Startup-MaxClients	Value of the Maximum Clients per Server (-Ma) startup parameter.
_Startup-MaxServers	Value of the Maximum Servers (-Mn) startup parameter.
_Startup-MaxUsers	Value of the Maximum Users (-n) startup parameter.
_Startup-Spin	Number of spinlock tries before time-out occurs.

Table 20–34: Index and Table Statistics Range (_StatBase)

Record	Description
_IndexBase	Index statistics base.
_TableBase	Table statistics base.

Table 20–35: Table Statistics File (_TableStat)

Record	Description
_TableStat-Create	Number of times create access has occurred to the table.
_TableStat-Delete	Number of times delete access has occurred to the table.
_TableStat-Read	Number of times read access has occurred to the table.
_TableStat-Update	Number of times update access has occurred to the table.

Table 20–36: Transaction File (_Trans)

Record	Description
_Trans-Coord	Name of the coordinator database.
_Trans-CoordTx	Current coordinator transaction number.
_Trans-Counter	Transaction count.
_Trans-Duration	Number of seconds the transaction required to complete.
_Trans-Flags	Transaction flags.
_Trans-Misc	Miscellaneous information.
_Trans-Num	Transaction number.
_Trans-State	Transaction state.
_Trans-Txtime	Transaction start time.
_Trans-Usrnum	User number of the user running the distributed transaction.

Table 20–37: Transaction End Lock Statistics (_TxeLock)

Record	Description
_Txe-Locks	Total number of times that transaction end lock is requested.
_Txe-Lockss	Number of concurrently held locks. (Only valid for update and commit locks.)
_Txe-Time	Not yet implemented; always 0.
_Txe-Type	Type of transaction end lock, such as update or commit.
_Txe-Waits	Number of times a transaction end lock request has been queued; how many times you had to wait.
_Txe-Waitss	Number of concurrently queued locks. (Only valid for update and commit locks.)

Table 20–38: Database Input/Output File (_UserIO) *(1 of 2)*

Record	Description
_UserIO-AiRead	Number of AI read operations performed by the process.
_UserIO-AiWrite	Number of AI write operations performed by the process.
_UserIO-BiRead	Number of BI read operations performed by the process.
_UserIO-BiWrite	Number of BI write operations performed by the process.
_UserIO-DbAccess	Number of database access operations performed by the process.
_UserIO-DbRead	Number of database read operations performed by the process.

Table 20–38: Database Input/Output File (_UserIO) *(2 of 2)*

Record	Description
_UserIO-DbWrite	Number of database write operations performed by the process.
_UserIO-Name	User name of the process.
_UserIO-Usr	User number of the process.

Table 20–39: Record Locking Table File (_UserLock)

Record	Description
_UserLock-Chain	Type of chain should always be REC, the record lock chain.
_UserLock-Flags	Any of four possible types of flags: L (LIMBO lock), P (PURGED lock entry), Q (QUEUED lock request), and U (UPGRADE request).
_UserLock-Misc	Miscellaneous information.
_UserLock-Name	User name of the process owning the lock.
_UserLock-Recid	Record ID for the record being locked.
_UserLock-Type	One of the three lock types: EXCL (EXCLUSIVE-LOCK), SHR (SHARE-LOCK), or NOLK (NO-LOCK).
_UserLock-Usr	User number of the process owning the lock.

Table 20–40: User Status (_UserStatus) *(1 of 2)*

Record	Description
_UserStatus-Counter	Count blocks accessed, records moved, tables moved, or indexes compacted, etc.
_UserStatus-ObjectId	The table or index worked on.

Table 20–40: User Status (_UserStatus) *(2 of 2)*

Record	Description
_UserStatus-ObjectType	The type of object worked on.
_UserStatus-Operation	Name of the online utility for which status is being monitored.
_UserStatus-State	The state the utility is in.
_UserStatus-Target	The end value of the counter (if known).
_UserStatus-UserId	User number.

A

Progress Monitor R&D Options

The Progress Monitor (PROMON) utility lets you monitor database activity and performance. Chapter 19, “Database Administration Utilities,” documents the main PROMON options. In addition, PROMON provides advanced options for in-depth monitoring of Progress activity and performance. These options are called R&D options. This appendix describes the R&D options.

NOTE: The R&D options might change in a future release.

Accessing PROMON R&D Options

Follow these steps:

1 ♦ Start PROMON by entering the following command:

Operating System	Syntax
UNIX, NT	promon *db-name*

In these commands, *db-name* specifies the database you want to monitor.

The PROMON main menu appears:

```
PROGRESS MONITOR Version 9

Database: /usr/wrk/sports2000

1. User Control
2. Locking and Waiting Statistics
3. Block Access
4. Record Locking Table
5. Activity
6. Shared Resources
7. Database Status
8. Shut Down Database

T. Transactions Control
L. Resolve Limbo Transactions
C. Coordinator Information

M. Modify Defaults
Q. Quit

Enter your selection:
```

The R&D option does not appear on the main menu.

2 ♦ At the Enter your selection prompt, enter R&D to access the R&D option.

The R&D main menu appears:

```
01/19/99          PROGRESS MONITOR (R&D)
14:50             Main (Top) Menu
1. Status Displays ...
2. Activity Displays ...
3. Other Displays ...
4. Administrative Functions ...
5. Adjust Monitor Options

Enter a number, <return>, P, T, or X (? for help):
```

3 ♦ Choose one of the following options:

- **Status Displays** — Shows information about the current state of the database and its users.
- **Activity Displays** — Shows information about database activity in the recent past.

  Activity displays shows the total number of each type of operation for the sample period (Total), the number of operations per minute (Per Min), the number of operations per second (Per Sec), and the number of operations per transaction (Per Tx).
- **Other Displays** — Shows information that does not fit into either the Status or Activity categories.
- **Administrative Functions** — Lets you see and change options related to administering a database.
- **Adjust Monitor Options** — Lets you change the way the monitor behaves.

Menus items followed by ellipses (`. . .`) access a lower level menu. See the "PROMON R&D Main Menu" section for the menus you can access from the PROMON R&D main (top) menu.

PROMON R&D Main Menu

Lets you access other menus by category. Table A–1 lists the main menu options.

Table A–1: PROMON R&D Main Menu *(1 of 3)*

1. Status Displays	Accesses a menu of Stats display options, including: 1. Database 2. Backup 3. Servers 4. Processes/Clients... 1. All Processes 2. Blocked Clients 3. Active Transactions 4. Local Interactive Clients 5. Local Batch Clients 6. Remote Clients 7. Background Processes 5. Files 6. Lock Table 7. Buffer Cache 8. Logging Summary 9. BI Log 10. AI Log 11. Two-Phase Commit 12. Startup Parameters 13. Shared Resources 14. Shared Memory Segments

Table A–1: PROMON R&D Main Menu *(2 of 3)*

2. Activity Displays	Accesses a menu of Activity display options, including: 1. Summary 2. Servers 3. Buffer Cache 4. Page Writers 5. BI Log 6. AI Log 7. Lock Table 8. I/O Operations by Type 9. I/O Operations by File 10. Space Allocation 11. Index 12. Record 13. Other
3. Other Displays	Accesses a menu of miscellaneous display options, including: 1. Performance Indicators 2. I/O Operations by Process 3. Lock Requests by User 4. Checkpoints
4. Administrative Functions	Accesses a menu of options that let you view and change settings, including: 1. Check Active Transaction Status 2. Check Two-phase Transactions 3. Resolve Limbo Transactions 4. Adjust Latch Options 5. Adjust Page Writer Options

Table A–1: PROMON R&D Main Menu *(3 of 3)*

5. Adjust Monitor Options	Accesses a menu of options that let you change PROMON monitor behavior, including: 1. Display Page Length 2. Clear Screen for First Page 3. Monitor Sampling Interval 4. Pause Between Displays 5. Pause Between Screens 6. Number of Auto Repeats 7. Change working area

Menu Options

At the bottom of each PROMON R&D screen, PROMON displays a prompt similar to the following:

```
Enter a number, <return>, P, T, or X (? for help):
```

The options you can choose depend on the screen you are viewing.

Options Available From All Screens

All screens provide the following options:

number

Displays the screen for the menu item with the number you specify.

<return>

Displays the next page of the current display. If there is no next page, returns to the previous menu.

P

Returns to the previous menu.

T

Returns to the main menu.

X

Exits the Progress Monitor utility.

?

Displays help information for each available option.

Options Available From Status Screens

At the bottom of each screen in the Status category, PROMON displays the following prompt:

```
Enter <return>, R, P, T, or X (? for help):
```

These options perform the following functions:

<return>

Displays the next page of the current display. If there is no next page, returns to the previous menu.

R

Repeats the current display.

P

Returns to the previous menu.

T

Returns to the main menu.

X

Exits the Progress Monitor utility.

?

Displays help information for each available option.

Options Available From Activity Screens

At the bottom of each screen in the Activity category, PROMON displays the following prompt:

```
Enter <return>, A, L, R, S, U, Z, P, T, or X (? for help):
```

These options perform the following functions:

<return>

Displays the next page of the current display. If there is no next page, returns to the previous menu.

A

Activates auto-repeat mode. The display restarts after the number of seconds specified by the current display pause time. (You can change the display pause time with the Adjust Monitor Options menu.) The display stops after the number of times specified by the auto repeat count or when you press **CTRL-C**. (You can change the auto-repeat count with the Adjust Monitor Options menu.)

L

Loads activity counters from start of the session.

R

Repeats the current display.

S

Displays activity data collected during the last sample interval only. (You can change the sample interval with the Adjust Monitor Options menu.) The display restarts after each sample interval.

U

Updates activity counters and restarts the display. The display shows only changes that have occurred since the initial set of data was collected.

Z

Zeros the activity counters. Subsequent updates show changes that have occurred since you chose this option.

P

Returns to the previous menu.

T

Returns to the main menu.

X

Exits the Progress Monitor utility.

?

Displays help information for each available option.

Status Displays→ Database

Displays general database status information. Figure A–1 shows a sample Database Status display.

```
01/25/02            Status: database
11:01:01

Database was started at:          01/25/02 9:17
It has been up for:               1:43:37
Database state:                   Open (1)
Database damaged flags:           None
Integrity flags:                  None
Most recent database open:        01/25/02 9:17
Previous database open:           01/25/02 9:17
Local cache file time stamp:      02/15/01 21:41
Database block size:              1024 bytes
Number of blocks allocated:       240 (240kb in area 7)
Empty blocks:                     9 (0% in area 7)
Free blocks:                      0 (0% in area 7)
RM Blocks with Free Space:        1 (0% in area 7)
Last transaction id:              399
Highest table number defined:     9
Database version number:          78
Shared memory version number:     7317
```

Figure A–1: PROMON Database Status Display

The display in Figure A–1 contains the following fields:

- **Database state** — The current operating mode of the database. Table A–2 describes the possible states.

Table A–2: Database States *(1 of 2)*

State	Description
Open	The database is open.
Not modified since last open	The database is closed.
Recovery	The database is performing crash recovery.

Table A–2: Database States *(2 of 2)*

State	Description
Index repair	The PROUTIL IDXBUILD utility is running to rebuild indexes.
Restore	The database is being restored.

- **Database damaged flags** — The state of the database. Table A–3 lists the possible flags.

Table A–3: Database Damaged Flags

State	Description
None	Normal.
Opened with -F	Crash recovery skipped. The database might be damaged.
Crashed with -i	Crashed without integrity. The database is damaged.
Crashed with -r	Crashed with buffered I/O. The database might be damaged.

- **Integrity flags** — The integrity mode in which the database is running. Table A–4 lists the possible flags.

Table A–4: Integrity Flags

State	Description
None	Normal.
Executing with -i	A crash in this state will damage the database.
Executing with -r	A crash in this state might damage the database.

- **Most recent database open** — The date and time when the broker for this database was started.
- **Previous database open** — The date and time when the database was previously started.

- **Local cache file time stamp** — The time stamp used to check the validity of the schema cache file. For more information, see the description of the Schema Cache File (`-cache`) parameter in Chapter 18, "Database Startup Parameters."
- **Database block size** — The database block size in bytes.
- **Number of blocks allocated** — The total number of blocks allocated to the database.
- **RM Blocks with free space** — The total number of blocks in the RM chain.
- **Highest table number defined** — The number of tables defined in the database.
- **Database version number** — Identifies the structures of all database-related data that is resident on disk. Progress uses this number to match different executable versions with the correct database structures.
- **Shared memory version number** — Identifies the version of all data structures that are resident in shared memory and used by the database manager. Progress uses this number to ensure that all executables accessing shared memory agree on what data is kept in shared memory and how that data is accessed. The shared-memory version number changes more frequently than the database version number.

Status Displays→ Backup

Use this option to display information about database backups. Figure A–2 shows a sample Backup Status display.

```
01/25/02            Status: backup
16:07

Most recent full backup:            01/25/02
Most recent incremental backup:     Never
Database changed since backup:      Yes
Sequence of last incremental:       0

Enter <return>, P, T, or X (? for help):
```

Figure A–2: PROMON Backup Status Display

The display in Figure A–2 contains the following fields:

- **Most recent full backup** — The date of the most recent full backup.
- **Most recent incremental backup** — The date of the most recent incremental backup.
- **Database changed since backup** — Yes, if the database has been modified since the last backup. No, if the database has not changed.
- **Sequence of last incremental** — The count of incremental backups made since the last full backup.

Status Displays→ Servers

Displays status information about Progress servers running on the system. Figure A–3 shows a sample Servers Status display.

```
01/25/02            Status: Servers
16:07

Sv                                          Cur.      Max.      Port
No     Pid     Type     Protocol     Logins  Users     Users     Num

0     16140    Login    TCP          0       0         5         2052
1     0        Inactive              0       0         0         0
2     0        Inactive              0       0         0         0
3     0        Inactive              0       0         0         0
4     0        Inactive              0       0         0         0
```

Figure A–3: PROMON Servers Status Display

The display contains the following fields:

- **Type** — The server type. Table A–5 describes the possible types.

Table A–5: Server Types

State	Description
Broker	A broker process. A broker logs in clients and starts remote servers.
Auto	A server started automatically by the broker.
Manual	A server started manually (with the Manual Server (`-m2`) startup parameter) after the broker was started. You might start a server manually for debugging purposes, or on a system where a server cannot be started automatically.
Login	A secondary login broker. When a database network includes more than one protocol, a secondary login broker performs broker functions for the second protocol.
Inactive	Placeholder in the server table for servers that have not yet started.

- **Protocol** — The communications protocol used by the server and its clients. The protocol can be one of the following: TCP/IP, TLI, or DECNET.

Status Displays→ Processes/Clients→ All Processes

Displays status information about Progress processes. Figure A–4 shows a sample Processes/Clients Status display.

```
01/25/99          All Processes Status
16:09

Usr    Name    Type  Wait          Trans id   Login time
0      rgw     BROK MTX   0         0          01/25/99 08:31
1      rgw     BIW IDLE   3853      0          01/25/99 08:33
2      rgw     APW --     0         0          01/25/99 08:33
3      rgw     SELF BUF   0         2601       01/25/99 08:34
4      rgw     MON --     0         0          01/25/99 08:57
```

Figure A–4: PROMON Processes/Clients Status Display

The display contains the following fields:

- **Type** — The process type. Table A–6 lists the possible types.

Table A–6: Process Types *(1 of 2)*

State	Description
AIW	After-image writer.
APLS	Application server.
APW	Asynchronous page writer.
BAT	Batch user.
BKUP	Online backup.
BIW	Before-image writer.
BROK	Broker.
MON	Monitor.
REMC	Remote client.
RPLA	Replication agent.

Table A–6: Process Types *(2 of 2)*

State	Description
RPLS	Replication server.
SELF	Self-service client.
SERV	Server for remote clients.
SHUT	Shutdown (PROSHUT).
WDOG	Watchdog.

- **Wait** — The wait type. Table A–7 describes the possible types. If no value is displayed, the process is not waiting on any database-related events.

Table A–7: Process Wait Types *(1 of 3)*

Type	Description
Lock Waits	
REC	Row (record) lock.
SCH	Schema lock.
TRAN	Transaction commit.
Resource Waits	
AIRD	After-image buffer read from disk.
AIWR	After-image buffer write to disk.
BIRD	Before-image buffer read from disk.
BIWR	Before-image buffer write to disk.
BKEX	Exclusive lock on database buffer.
BKSH	Share lock on database buffer.
BKSI	Share lock with intent to update on database buffer.

Table A–7: Process Wait Types *(2 of 3)*

Type	Description
DBBK	Database buffer being backed up.
DBRD	Database buffer read from disk.
DBWR	Database buffer write to disk.
RGET	Row get lock.
TXE	Transaction commit lock.
Latch Waits	
AIB	After-image buffer latch.
BFP	Buffer pool latch.
BHT	Buffer pool hash table latch.
BUF	Buffer pool buffer latch.
CPQ	Checkpoint queue latch.
DLC	Index delete chain.
GST	Global storage pool latch.
IXD	Index delete chain latch.
LKF	Lock table free chain latch.
LKP	Lock table purge chain latch.
LKT	Lock table latch.
LRS	Secondary LRU chain (not currently implemented).
LRU	Buffer pool LRU chain latch.
MTX	Log space allocation latch.
PWQ	Page writer queue latch.
SCH	Shared schema cache latch.

Table A–7: Process Wait Types *(3 of 3)*

Type	Description
SEQ	Sequence cache latch.
TXQ	Transaction queue latch.
TXT	Transaction table latch.
USR	User table latch.

Status Displays→ Processes/Clients→ Blocked Clients

Displays status information about clients waiting for a database-related resource. Figure A–5 shows a sample Blocked Clients Status display.

```
01/25/02       Status Blocked Clients
16:10

Usr  Name   Type    Wait      Trans id    Login time
2    eadams SELF    REC   3937       0    11/22/01 14:51
```

Figure A–5: PROMON Blocked Clients Status Display

The display contains the following fields:

- **Type** — The process type. Table A–6 lists the possible types.
- **Wait** — The resource for which the process is waiting.
- **Trans id** — The unique internal transaction number the process is currently working on.
- **Login time** — The time when the process logged into the database.

Status Displays→ Processes/Clients→ Active Transactions

Displays status information about active transactions. Figure A–6 shows a sample Active Transactions Status display.

```
01/25/02        Active Transactions Status
16:10

Usr   Name   Type    Login time    Tx Start Time   Trans id   Trans State

  0   rgw    SELF    01/25/02 08:31                    2601    Begin
```

Figure A–6: PROMON Active Transactions Status Display

The display contains the following fields:

- **Type** — The process type. Table A–6 lists the possible types.
- **Trans id** — The transaction ID.
- **Trans State** — The state of the transaction. Table A–8 lists the possible states.

Table A–8: Transaction States

State	Description
Begin	A transaction table entry was allocated, and a start record is being logged.
Active	The transaction is doing forward processing.
Prep	The transaction is preparing to enter phase 1 (ready to commit), but has not sent a ready-to-commit reply.
Phase 1	In phase 1, ready to commit.
Phase 2	In phase 2.
C	With two-phase commit, this user is logging for the coordinator for the transaction.
R	Ready to commit.
L	Limbo transaction.

Status Displays→ Processes/Clients→ Local Interactive Clients

Displays status information about local interactive client processes. Figure A–7 shows a sample Local Interactive Clients status display.

```
01/25/02            Local Interactive Clients Status
16:10

Usr   Name   Type Wait          Trans id   Login time     Pid

  0   rgw     BROK MTX    0            0   01/25/02 08:31    16147
```

Figure A–7: PROMON Local Interactive Clients Status Display

The display contains the following fields:

- **Usr** — The user number of the process.
- **Name** — The user name of the process.
- **Type** — The process type. Table A–6 lists the possible types.
- **Wait** — The wait type. Table A–7 describes the possible types.
- **Trans id** — The user's transaction ID.
- **Login time** — The time when the process logged into the database.
- **Pid** — The process ID number.

Status Displays→ Processes/Clients→ Local Batch Clients

Displays status information about local batch client processes. Figure A–8 shows a sample Local Batch Clients Status display.

```
01/25/02            Local Batch Clients Status
16:10

Usr   Name   Type Wait          Trans id   Login time        Pid

0     guy    SELF MTX      0         402   06/25/01 08:31   678
0     rgw    SELF REC      3937        0   06/25/01 08:36   687
```

Figure A–8: PROMON Local Batch Clients Status Display

The display contains the following fields:

- **Usr** — The user number of the process.
- **Name** — The user name of the process.
- **Type** — The process type. Table A–6 lists the possible types.
- **Wait** — The wait type. Table A–7 describes the possible types.
- **Trans id** — The user's transaction ID.
- **Login time** — The time when the process logged into the database.
- **Pid** — The process ID number.

Status Displays→ Processes/Clients→ Remote Clients

Displays status information about remote client processes. Figure A–9 shows a sample Remote Clients Status display.

```
01/25/02           Remote Clients Status
16:18

Usr   Name      Type Wait          Trans id      Login time       Serv

0     rgw       BROK MTX      0    0            01/25/01 08:31      0
```

Figure A–9: PROMON Remote Clients Status Display

The display contains the following fields:

- **Usr** — The user number of the process.
- **Name** — The user name of the process.
- **Type** — The process type. Table A–6 lists the possible types.
- **Wait** — The wait type. Table A–7 describes the possible types.
- **Trans id** — The user's transaction ID.
- **Login time** — The time when the process logged into the database.
- **Serv** — The server number serving the remote client.

Status Displays→ Processes/Clients→ Background Processes

Displays status information about background database processes. Figure A–10 shows a sample Background Processes Status display.

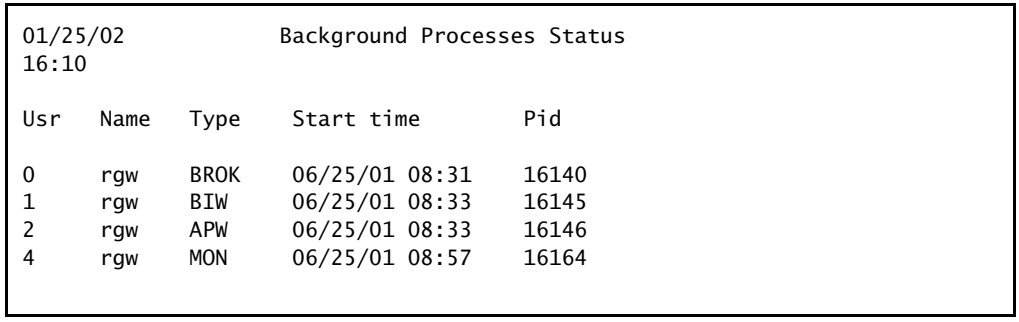


Figure A–10: PROMON Background Processes Status Display

The display contains the following fields:

- **Usr** — The user number of the process.
- **Name** — The user name of the process.
- **Type** — The process type. Table A–6 lists the possible types.
- **Start time** — The time when the process logged into the database.
- **Pid** — The process ID number.

Status Displays→ Files

Displays status information about Progress database files. Figure A–11 shows a sample Files Status display.

```
01/25/02           Status: Files
16:20
                   Size  Extend
File name          (KB)  (KB)

/disk1/sports.ai     288     24
/disk1/sports.db  245760  16384
/disk1/sports.bi  524288  65536

Enter <return>, P, T, or X (? for help):
```

Figure A–11: PROMON Files Status Display

The display contains the following fields:

- **File name** — The name of the file, including the path specification.
- **Size** — The file size, in kilobytes.
- **Extend** — The size of the last file extension, in kilobytes.

Status Displays→ Lock Table

Displays status information about the lock table. For a description of the lock table and its functions, see Chapter 14, "Managing Performance." Figure A–12 shows a sample Lock Table Status display.

```
01/25/02        Status: Lock Table
16:22

Usr    Name    Type    RECID    Flags

3      rgw     REC     7843     S
```

Figure A–12: PROMON Lock Table Status Display

The display contains the following fields:

- **Usr** — The user number of the user who owns the lock entry.
- **Name** — The user name.
- **Type** — The lock type (for example, REC, RGET).
- **RECID** — The RECID of the record and lock table entry.
- **Flags** — Table A–9 lists the possible flags.

Table A–9: Lock Table Flags *(1 of 2)*

Flag	Description
S	Share lock.
X	Exclusive lock.
U	Lock was upgraded from a share lock to an exclusive lock.
L	Limbo. Lock is being held until the transaction commits.
Q	Request is queued, and the user is waiting for a conflicting lock held by another user.

Table A–9: Lock Table Flags *(2 of 2)*

Flag	Description
P	Purged.
H	Hold.

Status Displays→ Buffer Cache

Display status information about Progress buffers. Figure A–13 shows a sample Buffer Cache Status display.

```
01/25/02              Status: Buffer Cache
16:21

Total buffers:                   170
Hash Table Size:                 43
Used Buffers:                    57
Empty Buffers:                   113
On lru Chain:                    168
On apw queue:                    0
On ckp queue:                    0
Modified buffers:                170
Marked for ckp:                  0
Last checkpoint number:          0
```

Figure A–13: PROMON Buffer Cache Status Display

The display contains the following fields:

- **Total buffers** — The total number of buffers in the buffer cache.
- **Hash Table Size** — The size of the buffer hash table.
- **Used Buffers** — The number of buffers currently in use.
- **Empty Buffers** — The number of empty buffers.
- **On lru Chain** — The number of buffers on the least recently used (LRU) chain.
- **On apw queue** — The number of buffers on the page writer queue.
- **On ckp queue** — The number of buffers on the checkpoint queue.
- **Modified buffers** — The number of dirty (modified) buffers.
- **Marked for ckp** — The number of buffers currently marked for checkpoint.
- **Last checkpoint number** — The most recent checkpoint number. As checkpoints begin, they are assigned a sequential number from the start of the session. The number is also the number of checkpoints that have occurred.

Status Displays→ Logging Summary

Displays status information for the database logging function. Figure A–14 shows a sample Logging Summary Status display.

```
01/25/02        Status: Logging Summary
16:22

Crash protection:                Yes
Delayed Commit:                  3 seconds
Before-image I/O:                Reliable
Before-image cluster age time:   60 seconds
BI Writer status:                Executing
Two-Phase Commit:                *** OFF ***
After-image journalling:         *** OFF ***
```

Figure A–14: PROMON Logging Summary Status Display

The display contains the following fields:

- **Crash protection** — Yes, if crash protection is enabled; No, if it is not.
- **Delayed Commit** — The current value of the Delayed BI File Write (`-Mf`) parameter.
- **Before-image I/O** — One of the following:
  - Reliable if synchronous or direct I/O is being used for BI writes.
  - BUFFERED if buffered I/O is being used for BI writes. Buffered I/O is not recommended.
- **Before-image cluster age time** — The period of time that must pass before Progress reuses a BI cluster. This period ensures that database blocks flushed at checkpoint are moved from the UNIX buffers on disk. When this occurs, the transaction is recorded on disk.
- **BI Writer status** — Executing, if a BIW is running.
- **Two-Phase Commit** — ON, if two-phase commit is enabled; OFF, if it is not.
- **After-image journalling** — ON, if after-imaging is enabled; OFF, if it is not.

Status Displays→ BI Log

Displays status information for before-image logging. Figure A–15 shows a sample BI Log Status display.

```
01/25/02       Status:BI Log
16:23

Before-image cluster age time:                  60 seconds
Before-image block size:                        1024 bytes
Before-image cluster size:                      128 kb (131072 bytes)
Number of before-image extents:                 0
Before-image log size (kb):                     512
Bytes free in current cluster:                  1548
Last checkpoint was at:                         01/25/02 13:56
Number of BI buffers:                           5
Full buffers:                                   0
```

Figure A–15: PROMON BI Log Status Display

The display contains the following fields:

- **Before-image cluster age time** — The period of time that must pass before Progress reuses a BI cluster. This period ensures that database blocks flushed at checkpoint are moved from the UNIX buffers on disk. When this occurs, the transaction is durably recorded on disk.
- **Before-image block size** — The BI block size.
- **Before-image cluster size** — The BI cluster size.
- **Number of before-image extents** — The number of BI files.
- **Before-image log size (kb)** — The size of the BI log file, in kilobytes.
- **Bytes free in current cluster** — The number of free bytes remaining in the current BI cluster.
- **Last checkpoint was at** — The time of the last checkpoint.
- **Number of BI buffers** — The number of BI buffers.
- **Full buffers** — The number of full BI buffers.

Status Displays→ AI Log

Displays status information for after-image logging. Figure A–16 shows a sample AI Log Status display.

```
04/14/02           Status: AI Log
16:53:30

After-image begin date:            04/14/02 16:27
After-image new date:              04/14/02 16:27
After-image open date:             04/14/02 16:48
After-image generation number:     0
Number of after-image extents:     0
Number of AI buffers:              1
After-image block size:            1024 bytes
After-image log size:              15K
```

Figure A–16: PROMON AI Log Status Display

The display contains the following fields:

- **After-image begin date** — The date of the last AIMAGE BEGIN command.
- **After-image new date** — The date of the last AIMAGE NEW command.
- **After-image open date** — The date when the AI log was last opened.
- **After-image generation number** — The generation number of the current AI extent.
- **Number of after-image extents** — The number of AI files or extents.
- **Number of AI buffers** — The number of buffers in the AI buffer pool. You can change the number of AI buffers with the After-image Buffers (`-aibufs`) startup parameter. The default value is 1.
- **After-image block size** — The size of the after-image block. You can change the AI block size. This reduces I/O rates on disks where AI files are located. For more information, see Chapter 14, "Managing Performance."
- **After-image log size** — The size of the AI log file, in kilobytes.

Status Displays→ Two-phase Commit

Displays status information about two-phase commit. Figure A–17 shows a sample Two-Phase Commit Status display.

```
04/14/02             Status: Two-Phase Commit
17:02:19

Coordinator nickname:    mydemo
Coordinator priority:    0
After-image journalling:    Yes
```

Figure A–17: PROMON Two-phase Commit Status Display

The display contains the following fields:

- **Coordinator nickname** — The nickname that Progress uses to identify the coordinator database. You specify a nickname when you enable two-phase commit.
- **Coordinator priority** — The priority for the coordinator database. You specify the priority when you enable two-phase commit.
- **After-image journalling** —Yes, if after-imaging is enabled; No, if it is not. Progress Software recommends you use after-imaging if you use two-phase commit.

Status Displays→ Startup Parameters

Displays values of Progress database startup parameters. Figure A–18 shows a sample Startup Parameters Status display.

```
01/25/02       Status: Startup Parameters
16:23

Maximum clients:                        21
Maximum servers:                        4
Maximum clients per server:             0
Lock table size:                        512 entries
Database buffers:                       168 (168 kb)
APW queue check time:                   500 milliseconds
APW scan time:                          1 seconds
APW buffers to scan:                    1
APW max writes/scan:                    25
Before-image buffers:                   5 (5 kb)
After-image file name:                  -
After-image buffers:                    1 (1 kb)
```

Figure A–18: PROMON Startup Parameters Status Display

The display contains the following fields:

- **Maximum clients** — The value of the Number of Users (`-n`) parameter.
- **Maximum servers** — The value of the Maximum Servers (`-mn`) parameter.
- **Maximum clients per server** — The value of the Maximum Clients per Server (`-Ma`) parameter.
- **Lock table size** — The value of the Lock Table Entries (`-L`) parameter.
- **Database buffers** — The value of the Blocks in Database Buffers (`-B`) parameter.
- **APW queue check time** — The value of the Page Writer Queue Delay (`-pwqdelay`) parameter.
- **APW scan time** — The value of the Page Writer Scan Delay (`-pwsdelay`) parameter.

- **APW buffers to scan** — The value of the Page Writer Scan (`-pwscan`) parameter.
- **APW max writes/scan** — The value of the Page Writer Maximum Buffers (`-pwwmax`) parameter.
- **Before-image buffers** — The value of the Before-image Buffers (`-bibufs`) parameter.
- **After-image file name** — The value of the After-image Filename (`-a`) parameter.
- **After-image buffers** — The value of the After-image Buffers (`-aibufs`) parameter.

Status Displays→ Shared Resources

Displays status information about shared database resources. Figure A–19 shows a sample Shared Resources Status display.

```
01/25/02          Status: Shared Resources
16:24

Active transactions:                      1
Lock table entries in use:                1 of 512
Lock table high water mark:               2
Number of servers:                        0 (5 allocated)
Total clients:                            1 (21 allocated)
   Self service:                          1
   Remote:                                0
   Batch:                                 0
Watchdog status:                          *** Not executing ***
BIW status:                               *** Not executing ***
Number of page writers:                   1
Number of monitors:                       1
Number of semaphores allocated:           29
Shared memory allocated:                  400 K (1 segments)
```

Figure A–19: PROMON Shared Resources Status Display

The display contains the following fields:

- **Active transactions** — The number of currently active transactions.
- **Lock table entries in use** — The number of lock table entries that are currently being used, and the total number of entries available.
- **Lock table high water mark** — The maximum number of lock table entries that were in use simultaneously.
- **Number of servers** — The number of servers running.
- **Total clients** — The number of clients, broken down by the following types: Self service, Remote, and Batch.
- **Watchdog status** — Executing, if the Watchdog process is executing; Not executing, if the Watchdog is not executing.
- **BIW status** — Executing, if the BIW is executing; Not executing, if the BIW is not executing.

- **Number of page writers** — The number of APWs running.
- **Number of monitors** — The number of Progress Monitor processes running.
- **Number of semaphores allocated** — The number of UNIX semaphores allocated.
- **Shared memory allocated** — The amount of shared memory allocated, in kilobytes and in segments. For more information an shared-memory allocation, see Chapter 14, “Managing Performance.”

Status Displays→ Shared Memory Segments

Displays general information about shared-memory segments allocated to the Progress database. Figure A–20 shows a sample Shared Memory Segments Status display.

```
Seg     Id     Size     Used     Free

1       400    303104   152320   150784
```

Figure A–20: PROMON Shared Memory Segments Status Display

The display contains the following fields:

- **Seg** — The segment number. Each segment receives a sequential number when it is created.
- **Id** — The number the operating system uses to reference this segment.
- **Size** — The total allocated segment size, in bytes.
- **Used** — The amount of shared memory used from the segment, in bytes.
- **Free** — The amount of shared memory allocated but not yet used in the segment, in bytes.

Activity Displays→ Summary

Displays general information about database activity. Figure A–21 shows a sample Summary Activity display.

```
01/25/02       Activity: Summary
16:27          from 01/25/02 15:31 to 01/25/02 16:04 (2 min. 48 sec.)

Event           Total   Per Sec.    Event          Total   Per Sec.

Commits          1208       0.7     DB Reads          57       0.0
Undos               0       0.0     DB Writes          3       0.0
Record Reads     1334       0.8     BI Reads          13       0.0
Record Updates   1208       0.7     BI Writes        131       0.0
Record Creates   0          0.0     AI Writes          0       0.0
Record Deletes   0          0.0     Checkpoints        0       0.0
Record Locks     3625       2.3     Flushed at chkpt   0       0.0
Record Waits        0       0.0

Rec Lock Waits  0%    BI Buf Waits     0%     AI Buf Waits   0%
Writes by APW   0%    Writes by BIw    96%    Writes by AIW  0%
DB Size:        352K  BI size:         512K   AI Size:       OK
Empty blocks:   10    Free blocks:     0      RM chain:      1
Buffer hits:    98%   Active trans:    1


0 Servers, 1 Clients (1 Local, 0 Remote, 0 Batch), 1 APWs
```

Figure A–21: PROMON Summary Activity Display

The Summary Activity display shows the number of events that have occurred on the system, including the cumulative total and the number of events per second. The display includes the following events:

- **Commits** — The number of transactions all users have committed.
- **Undos** — The number of transactions rolled back.
- **Record Reads** — The number of records read.
- **Record Updates** — The number of records updated.
- **Record Creates** — The number of records created.
- **Record Deletes** — The number of records deleted.

- **Record Locks** — The number of record locks used.
- **Record Waits** — The number of times users have waited to access a locked record.
- **DB Reads** — The number of database blocks read.
- **DB Writes** — The number of database blocks written to disk.
- **BI Reads** — The number of BI blocks read.
- **BI Writes** — The number of BI blocks written to disk.
- **AI Writes** — The number of AI blocks written to disk.
- **Checkpoints** — The number of checkpoints that have been performed.
- **Flushed at chkpt** — The number of database buffers that have been flushed to disk because they were not written by the time the checkpoint ended.

The Summary Activity display also shows the following information:

- **Rec Lock Waits** — The percentage of record accesses that result in record lock waits. A record lock wait occurs when Progress must wait to access a locked record.

  For optimum performance, try to keep this number as low as possible. You can lower this number by modifying your application to perform shorter transactions so that record locks are held for shorter periods of time.

- **BI Buf Waits** — The percentage of BI buffer waits. A BI buffer wait occurs when Progress must wait to access a BI buffer.

  For optimum performance, try to keep this number as low as possible. To decrease this number, allocate more BI buffers with the Before-image Buffers (`-bibufs`) parameter.

- **AI Buf Waits** — The percentage of AI buffer waits. An AI buffer wait occurs when Progress must wait to access an AI buffer.

  For optimum performance, try to keep this number as low as possible. To decrease this number, allocate more AI buffers with the After-image Buffers (`-aibufs`) parameter.

- **Writes by APW** — The percentage of database blocks written to disk by the APW; this is a percentage of the total number of database blocks written by Progress.

  For optimum performance, try to keep this number as high as possible. To increase this number, start more APWs and increase the cluster size.

- **Writes by BIW** — The percentage of BI blocks written to disk by the BIW; this is a percentage of the total number of BI blocks written to disk by Progress.

  For optimum performance, try to keep this percentage fairly high. You can increase the number of BI buffers with the `-bibufs` parameter.

- **Writes by AIW** — The percentage of AI blocks written to disk by the AIW; this is a percentage of the total number of AI blocks written by Progress.

  For optimum performance, try to keep this percentage fairly high. You can increase the number of AI buffers with the `-aibufs` parameter.

- **DB Size** — The size of your database, in kilobytes.

- **BI Size** — The size of your BI file, in kilobytes.

- **AI Size** — The size of your AI file, in kilobytes.

- **Empty blocks** — The number of empty (never used) database blocks.

- **Free blocks** — The number of blocks on the database's free chain. The free chain is a chain of previously used and then deallocated database blocks.

- **RM chain** — The number of blocks on the database's RM chain. The RM chain is a chain of partially filled database blocks.

- **Buffer hits** — The percentage of buffer hits. A buffer hit occurs when Progress locates a database record in the Progress buffer pool and does not have to read the record from disk.

  For optimum performance, keep this number as high as possible. To increase this number, allocate more buffers with the `-B` parameter. Increase the number of buffers until one of the following occurs:

  – The number of I/Os per second is reduced to less than half the capacity of the drives.

  – The buffer hit rate stops improving.

  – The system starts paging.

- **Active trans** — The number of active transactions.

The last line of the Summary Activity display summarizes the current number of each type of process running against the database at the time you run the Activity option. Table A–10 defines the process types.

Table A–10: Process Types

Field	Description
Servers	Number of servers running against your database. This is the current value at the time you run the Activity option, not a cumulative total.
Clients	Number of client sessions running. This field is further divided into the type of user: Local, Remote, or Batch. This is the current value at the time you run the Activity option, not a cumulative total.
APWs	Number of APWs running against your database. This is the current value at the time you run the Activity option, not a cumulative total.

Activity Displays→ Servers

Displays information about server activity. Figure A–22 shows a sample Servers Activity display.

```
01/25/02              Activity: Servers
16:28                 from 01/25/02 13:56 to 01/25/02 13:57 (19 sec)

                         Total       Per Min      Per Sec     Per Tx
Messages received          204           131         2.19       0.00
Messages sent              152            98          1.6       0.00
Bytes received           13252          8549       142.49       0.00
Bytes sent               40683         26247       437.45       0.00
Records received             0             0         0.00       0.00
Records sent               122            79         1.31       0.00
Queries received            54            35         0.58       0.00
Time slices                 51            32         0.54       0.00

Activity for Server
```

Figure A–22: PROMON Servers Activity Display

The display contains the following fields:

- **Messages received** — The number of network packets received by the server.
- **Messages sent** — The number of network packets sent by the server.
- **Bytes received** — The number of bytes received by the server.
- **Bytes sent** — The number of bytes sent by the server.
- **Records received** — The number of transferred records received.
- **Records sent** — The number of transferred records sent.
- **Queries received** — The number of query requests received.
- **Time slices** — The number of query time slice switches.

Activity Displays→ Buffer Cache

Displays activity information about the database buffer cache (also called the buffer pool). Figure A–23 shows a sample Buffer Cache Activity display.

```
01/25/02              Activity: Buffer Cache
16:28                 from 01/25/02 13:56 to 01/25/02 13:57 (19 sec)

                         Total       Per Min      Per Sec     Per Tx
Logical Reads            23          73           1.21        0.00
Logical Writes           0           0            0.00        0.00
O/S reads                8           25           0.42        0.00
O/S writes               2           6            0.10        0.00
Checkpoints              0           0            0.00        0.00
Marked at checkpoint     0           0            0.00        0.00
Flushed at checkpoint    0           0            0.00        0.00
Writes deferred          0           0            0.00        0.00
LRU skips                0           0            0.00        0.00
LRU writes               0           0            0.00        0.00
APW Enqueues             0           0            0.00        0.00

Hit Ratio: 68 %
```

Figure A–23: PROMON Buffer Cache Activity Display

The display lists the following types of buffer cache requests:

- **Logical Reads** — The number of client requests for database block read operations.
- **Logical Writes** — The number of client requests for database block write operations.
- **O/S reads** — The number of database blocks read from disk.
- **O/S writes** — The number of database block writes to disk.
- **Checkpoints** — The number of checkpoint operations.
- **Marked at checkpoint** — The number of blocks scheduled to be written before the end of a checkpoint.
- **Flushed at checkpoint** — The number of blocks that were not written during the checkpoint and that had to be written all at once at the end of the checkpoint.
- **Writes deferred** — The total number of changes to blocks that occurred before the blocks were written. Each deferred write is potentially an I/O operation saved.

- **LRU skips** — The number of times a buffer on the LRU chain was skipped because it was locked or modified.
- **LRU writes** — The number of blocks written to free a buffer for a read operation.
- **APW Enqueues** — The number of modified buffers placed on the APW queue for writing.
- **Hit Ratio** — The percentage of buffer cache requests that did not require a physical disk I/O operation.

Activity Displays→ Page Writers

Displays information about asynchronous page writer (APW) activity. Figure A–24 shows a sample Page Writers Activity display.

```
01/25/02             Activity: Page Writers
16:29                from 01/25/02 13:56 to 01/26/02 11:23 (21 hrs 27 min)

                                    Total   Per Min   Per Sec    Per Tx
Total DB writes                     3       0         0.00       0.00
APW DB writes                       0       0         0.00       0.00
    scan writes                     8       25        0.42       0.00
    APW queue writes                2       6         0.10       0.00
    ckp queue writes                0       0         0.00       0.00
    scan cycles                     0       0         0.00       0.00
    buffers scanned                 0       0         0.00       0.00
    bufs checkpointed               173     0         0.11       0.14
Checkpoints                         8211    0         5.22       6.79
Marked at checkpoint                0       0         0.00       0.00
Flushed at checkpoint               0       0         0.00       0.00

Number of APWs:                              1
```

Figure A–24: PROMON Page Writers Activity Display

The display lists the following types of APW operations:

- **Total DB writes** — The total number of database write operations performed by all processes.
- **APW DB writes** — The number of database write operations performed by the APW. This is a subset of the total number of DB writes:
  - **Scan writes** — The number of buffers written during the scan cycle.
  - **APW queue writes** — The number of buffers written to clear the APW queue.
  - **Ckp queue writes** — The number of buffers written from the checkpoint queue.
  - **Scan cycles** — The number of scan cycles. During a scan cycle, the APW scans a portion of the buffer pool to look for modified buffers.
  - **Buffers scanned** — The number of buffers scanned during each cycle.
  - **Bufs checkpointed** — The number of buffers checkpointed.

- **Checkpoints** — The number of checkpoints that have occurred.
- **Marked at checkpoint** — The number of buffers that were scheduled to be written before the end of the checkpoint.
- **Flushed at checkpoint** — The number of blocks that were not written during the checkpoint and had to be written all at once at the end of the checkpoint.
- **Number of APWs** — The number of APWs running.

Activity Displays→ BI Log

Displays information about BI Log Activity. Figure A–25 shows a sample BI Log Activity display.

```
01/25/02            Activity: BI Log
11:36:56            from 04/12/02 13:56 to 04/13/02 11:23 (21 hrs 27 min)

                                    Total   Per Min   Per Sec    Per Tx
Total BI writes                     131     0         0.08       0.10
BIW BI writes                       127     0         0.08       0.10
Records written                     3630    25        2.31       3.00
Bytes written                       129487  0         82.42      107.19
Total BI Reads                      13      0         0.00       0.01
Records read                        0       0         0.00       0.00
Bytes read                          0       0         0.00       0.00
Clusters closed                     0       0         0.00       0.00
Busy buffer waits                   0       0         0.00       0.00
Empty buffer waits                  0       0         0.00       0.00
Log force waits                     0       0         0.00       0.00
Partial Writes                      4       0         0.00       0.00
```

Figure A–25: PROMON BI Log Activity Display

The display lists the following types of BI operations:

- **Total BI writes** — The number of writes to the BI file.
- **BIW BI writes** — The number of writes to the BI file performed by the before-image writer (BIW). For good performance, this number should be high in relation to the total number of BI writes.
- **Records written** — The number of BI records (notes) written to the BI file.
- **Bytes written** — The amount of data written to the BI file, in bytes.
- **Total BI Reads** — The number of BI blocks read from the BI file (to undo transactions).
- **Records read** — The number of BI records (notes) read from the BI file.
- **Bytes read** — The amount of data read from the BI file, in bytes.
- **Clusters closed** — The number of BI clusters filled and closed in preparation for reuse.

- **Busy buffer waits** — The number of times a process had to wait for someone else to finish before being able to write to a BI file.
- **Empty buffer waits** — The number of times a process had to wait because all buffers were full.
- **Log force waits** — The number of waiting-for-commit records to be written to disk.
- **Partial Writes** — The number of writes to the BI file made before the BI buffer is full. This might happen if:
  - The Delayed BI File Write (`-Mf`) parameter timer expired before the buffer was filled.
  - An APW attempts to write a block whose changes are recorded in a BI buffer that has not been written. Because BI notes must be flushed before the AI note is flushed, the AIW writes the data in the BI buffer before the buffer is full so it can do the AI write.
  - An AIW ran ahead of the BIW. Because BI notes must be flushed before the AI notes can be written, the AIW writes the BI buffer before it is full, so it can do the AI write.

Activity Displays→ AI Log

Displays after-imaging activity. Figure A–26 shows a sample AI Log Activity display.

```
01/25/02            Activity: AI Log
11:36:56            from 04/12/02 13:56 to 04/13/02 11:23 (21 hrs 27 min)

                                    Total   Per Min   Per Sec    Per Tx
Total AI writes                     131     0         0.08       0.10
AIW AI writes                       127     0         0.08       0.10
Records written                     3630    25        2.31       3.00
Bytes written                       129487  0         82.42      107.19
Busy buffer waits                   13      0         0.00       0.01
Buffer not avail                    0       0         0.00       0.00
Partial Writes                      4       0         0.00       0.00
Log force waits                     0       0         0.00       0.00
```

Figure A–26: PROMON AI Log Activity Display

The display lists the following operations:

- **Total AI writes** — The total number of writes to the AI file.
- **AIW AI writes** — The number of AI writes performed by the after-image writer (AIW). This is a subset of the total AI writes.
- **Records written** — The number of records (notes) written to the AI file.
- **Bytes written** — The amount of AI data written to the AI file, in bytes.
- **Busy buffer waits** — The number of times a process had to wait because a buffer was held.
- **Buffer not avail** — The total number of times a process had to wait because a buffer was not available.

- **Partial writes** — The number of writes to the AI file made before the AI buffer is full. This might happen if:
  - The Delayed BI File Write (`-Mf`) parameter timer expired before the buffer was filled.
  - An APW attempts to write a block whose changes are recorded in an AI buffer that has not been written.
- **Log force waits** — The number of waiting-for-commit records to be written to disk.

Activity Displays→ Lock Table

Displays information about Lock Table Activity. The database engine stores record locks in the lock table. If a user tries to acquire a lock on a record, and the lock table overflows, the server aborts.

Figure A–27 shows a sample Lock Table Activity display.

```
01/25/02             Activity: Lock Table
16:24:02             from 01/25/02 15:31 to 01/25/02 16:24

                                    Total   Per Min   Per Sec    Per Tx

Share requests                      0       0         0.00       0.00
Exclusive requests                  0       0         0.00       0.00
Upgrade requests                    0       0         0.00       0.00
Rec Get requests                    0       0         0.00       0.00
Share grants                        0       0         0.00       0.00
Exclusive grants                    0       0         0.00       0.00
Upgrade grants                      0       0         0.00       0.00
Rec Get grants                      0       0         0.00       0.00
Share waits                         0       0         0.00       0.00
Exclusive waits                     0       0         0.00       0.00
Upgrade waits                       0       0         0.00       0.00
Rec Get waits                       0       0         0.00       0.00

Requests cancelled                  0       0         0.00       0.00
Downgrades                          0       0         0.00       0.00
Redundant requests                  0       0         0.00       0.00
```

Figure A–27: PROMON Lock Table Activity Display

The display lists the following operations:

- **Share requests** — The number of user requests for share locks.
- **Exclusive requests** — The number of user requests for exclusive locks.
- **Upgrade requests** — The number of requests to upgrade a lock from shared lock to exclusive lock.
- **Rec Get requests** — The number of granted record get locks.
- **Share grants** — The number of granted share lock requests.
- **Exclusive grants** — The number of granted exclusive lock requests.

- **Upgrade grants** — The number of granted requests for lock upgrades.
- **Rec Get grants** — The number of granted requests for record get locks.
- **Share waits** — The number of times processes waited for share lock.
- **Exclusive waits** — The number of times processes waited for an exclusive lock.
- **Upgrade waits** — The number of times processes waited for upgrades.
- **Rec Get waits** — The number of times processes waited for record gets.
- **Requests cancelled** — The number of lock requests that were cancelled.
- **Downgrades** — The number of locks downgraded.
- **Redundant requests** — The number of requests for a lock already held.

Activity Displays→ I/O Operations By Type

Displays information about I/O activity, organized by type. Figure A–28 shows a sample I/O Operations by Type Activity display.

```
01/25/02              Activity: I/O Operations by Type
16:31               from 01/25/02 15:31 to 01/25/02 16:04

                                    Total   Per Min   Per Sec    Per Tx
Database reads                      57      0         0.03       0.04
  Index blocks                      13      0         0.00       0.01
  Data blocks                       44      0         0.02       0.03
BI reads                            13      0         0.00       0.00
AI Reads                            0       0         0.00       0.05
Total reads                         70      0         0.04       0.05

Database writes                     3       0         0.00       0.00
  Index blocks                      0       0         0.00       0.00
  Data blocks                       3       0         0.08       0.00
BI writes                           131     0         0.00       0.10
AI writes                           0       0         0.00       0.00
Total writes                        134     0         0.08       0.11
```

Figure A–28: PROMON I/O Operations By Type Activity Display

The display lists the following types of I/O operations:

- **Database reads** — Total number of reads from the database:
  - **Index blocks** — The number of reads of index blocks.
  - **Data blocks** — The number of reads of data blocks.
- **BI reads** — The number of blocks read from BI files.
- **AI reads** — The number of blocks read from AI files.
- **Total reads** — Total number of database, BI, and AI read operations.

- **Database writes** — Total number of writes to the database:
  - **Index blocks** — The number of writes of index blocks.
  - **Data blocks** — The number of writes of data blocks.
- **BI writes** — The number of BI write operations.
- **AI writes** — The number of AI write operations.
- **Total writes** — Total number of database, BI, and AI write operations.

Activity Displays→ I/O Operations By File

Displays information about I/O activity, organized by file. Figure A–29 shows a sample I/O Operations by File Activity display.

```
01/25/02           Activity: I/O Operations by File
16:32              from 01/25/02 15:31 to 01/25/02 16:04 (for 32 min. 48 sec.)

                                Total      Per Min    Per Sec    Per Tx

/disk1/demo/.db
Reads                           8          48         0.80       0.00
Writes                          2          12         0.20       0.00
Extends                         0          1          0.00       0.00

                                Total      Per Min    Per Sec    Per Tx

/disk1/demo.bi
Reads                           13         78         1.30       0.00
Writes                          1          6          0.10       0.00
Extends                         0                     0.00       0.00

                                Total      Per Min    Per Sec    Per Tx

/disk1/demo.ai
Reads                           29                    0.03       0.04
Writes                          3                     0.00       0.00
Extends                         0                     0.00       0.00
```

Figure A–29: PROMON I/O Operations By File Activity Display

For each file, the display lists the following types of I/O operations:

- **Reads** — The number of read operations performed on the file.
- **Writes** — The number of write operations performed on the file.
- **Extends** — The number of extend operations performed on the file.

Activity Displays→ Space Allocation

Displays information about Space Allocation. Figure A–30 shows a sample Space Allocation Activity display.

```
01/25/02         Activity: Space Allocation
14:52:41         from 01/25/02 13:56 to 01/26/02 11:23 (for 21 hrs 27 min)

                                 Total      Per Min    Per Sec    Per Tx
Database extends                 57         0          0.03       0.04
Take free block                  13         0          0.00       0.01
Return free block                44         0          0.02       0.03
Alloc rm space                   13         0          0.00       0.00
Alloc from rm                    0          0          0.00       0.05
Alloc from free                  70         0          0.04       0.05
Bytes allocated                  3          0          0.00       0.00
rm blocks examined               0          0          0.00       0.00
Remove from rm                   3          0          0.00       0.00
Add to rm, front                  131        0          0.08       0.10
Add to rm, back                  0          0          0.00       0.00
Move rm front to back            134        0          0.08       0.11
Removed locked rm entry          0          0          0.00       0.00
```

Figure A–30: PROMON Space Allocation Activity Display

The display lists the following operations:

- **Database extends** — The number of times the database was extended.
- **Take free block** — The number of times a block was used from the free chain.
- **Return free block** — The number of times a block was returned to the free chain.
- **Alloc rm space** — The number of times space was allocated for a record or record fragment.
- **Alloc from rm** — The number of times space was allocated from the rm chain.
- **Alloc from free** — The number of times space was allocated from the free chain.
- **Bytes allocated** — The number of bytes allocated for record fragments.
- **rm blocks examined** — The number of blocks examined in the rm chain while looking for space for a record fragment.
- **Remove from rm** — The number of blocks removed from the rm chain.

- **Add to rm, front** — The number of blocks added to the front of the rm chain.
- **Add to rm, back** — The number of blocks added to the back of the rm chain.
- **Move rm front to back** — The number of blocks moved from the front to the back of the rm chain.
- **Remove locked rm entry** — The number of rm chain entries that were locked.

Activity Displays→ Index

Displays information about index activity. Figure A–31 shows a sample Index Activity display.

```
04/14/02          Activity: Index
17:04:35          from 04/14/02 17:01 to 04/14/02 17:01 (10 sec)

                                  Total       Per Min     Per Sec     Per Tx

Find index entry                  0           0           0.00        0.00
Create index entry                0           0           0.00        0.00
Delete index entry                0           0           0.00        0.00
Remove locked entry               0           0           0.00        0.00
Split block                       0           0           0.00        0.00
Free block                        0           0           0.00        0.00
```

Figure A–31: PROMON Index Activity Display

The display shows the following operations:

- **Find index entry** — The number of times an index entry was looked up.
- **Create index entry** — The number of new index entries generated.
- **Delete index entry** — The number of index entries deleted.
- **Remove locked entry** — The number of old locks released at transaction end.
- **Split block** — The number of block splits from adding indexes.
- **Free block** — The number of index blocks freed during deletes.

Activity Displays→ Record

Displays information about record activity. Figure A–32 shows a sample Record Activity display.

```
04/14/02          Activity: Record
17:04:35          from 04/14/02 17:01 to 04/14/02 17:01 (10 sec)

                                 Total      Per Min    Per Sec    Per Tx

Read record                      0          0          0.00       0.00
Update record                    0          0          0.00       0.00
Create record                    0          0          0.00       0.00
Delete record                    0          0          0.00       0.00
Fragments read                   0          0          0.00       0.00
Fragments created                0          0          0.00       0.00
Fragments deleted                0          0          0.00       0.00
Fragments updated                0          0          0.00       0.00
Bytes read                       2491       14946      129.10     0.00
Bytes created                    0          0          0.00       0.00
Bytes deleted                    0          0          0.00       0.00
Bytes updated                    0          0          0.00       0.00
```

Figure A–32: PROMON Record Activity Display

The display shows the following operations:

- **Read record** — The number of records read.
- **Update record** — The number of records modified.
- **Create record** — The number of records created.
- **Delete record** — The number of records deleted.
- **Fragments read** — The number of fragments read.
- **Fragments created** — The number of fragments created.
- **Fragments deleted** — The number of fragments deleted.
- **Fragments updated** — The number of fragments updated.

- **Bytes read** — The number of bytes read.
- **Bytes created** — The number of record bytes created.
- **Bytes deleted** — The number of record bytes deleted.
- **Bytes updated** — The number of record bytes updated.

Activity Displays→ Other

Displays information about miscellaneous activity. Figure A–33 shows a sample Other Activity display.

```
04/14/02          Activity: Other
17:04:35          from 04/14/02 17:01 to 04/14/02 17:01 (10 sec)

                                  Total       Per Min     Per Sec     Per Tx

Commit                            0           0           0.00        0.00
Undo                              0           0           0.00        0.00
Wait on semaphore                 0           0           0.00        0.00
Flush master block                3           18          0.30        0.00
```

Figure A–33: PROMON Other Activity Display

The display shows the following operations:

- **Commit** — The number of transactions committed.
- **Undo** — The number of transactions rolled back.
- **Wait on semaphore** — The number of times a process had to wait for a resource.
- **Flush master block** — The number of times the database master block was written to disk.

Other Displays→ Performance Indicators

Displays activity statistics related to performance. Figure A–34 shows a sample Performance Indicators display.

```
04/14/02         Activity: Performance Indicators
17:04:35         from 04/15/02 10:01 to 04/19/02 13:26 (99 hrs 24 min)

                                 Total      Per Min    Per Sec    Per Tx

Commits                          0          0          0.00       0.00
Undos                            0          0          0.00       0.00
Index Operations                 0          0          0.00       0.00
Record Operations                3          18         0.30       0.00
Total o/s i/o                    28         0          0.00       0.00
Total o/s reads                  23         0          0.00       0.00
Total o/s writes                 5          0          0.00       0.00
Background o/s writes            5          0          0.00       0.00
Partial log writes               1          0          0.00       0.00
Database extends                 0          0          0.00       0.00
Total waits                      0          0          0.00       0.00
Lock waits                       0          0          0.00       0.00
Resource waits                   0          0          0.00       0.00

buffer pool hit rate: 68%
```

Figure A–34: PROMON Performance Indicators Display

The Performance Indicators display shows the total number of each type of operation for the sample period (Total), the number of operations per minute (Per Min) the number of operations per second (Per Sec), and the number of operations per transaction (Per Tx). The display shows the following operations:

- **Commits** — The number of committed transactions.
- **Undos** — The number of transactions that were rolled back.
- **Index Operations** — The total of all operations performed on indexes (for example, index additions and deletions).
- **Record Operations** — The total of all operations on records (for example, record additions and deletions).
- **Total o/s i/o** — The total number of read and write operations performed.

- **Total o/s reads** — The total number of read operations performed.
- **Total o/s writes** — The total number of write operations performed.
- **Background o/s writes** — The total number of writes performed by background writers (APWs, BIW, and AIW). For more information on background writers, see Chapter 14, "Managing Performance."
- **Partial log writes** — Writes to the BI file made before the BI buffer is full. This might happen if:
  - The Delayed BI File Write (`-Mf`) parameter timer expired before the buffer was filled.
  - An APW attempts to write a block whose changes are recorded in a BI buffer that has not been written. Because BI notes must be flushed before the AI note is flushed, the AIW writes the data in the BI buffer before the buffer is full so it can do the AI write.
  - An AIW ran ahead of the BIW. Because BI notes must be flushed before the AI notes can be written, the AIW writes the BI buffer before it is full, so it can do the AI write.
- **Database extends** — The total number of times the database was made larger by allocating operating system blocks.
- **Total waits** — The total number of waits.
- **Lock waits** — The number of times Progress waited for a lock to be released.
- **Resource waits** — The number of times that Progress waited for a resource to become available, such as a row lock, a buffered lock in shared memory, or a transaction end lock.
- **Buffer pool hit rate** — The percentage of times that Progress found a record in the Progress buffer pool and did not have to read the record from disk.

Other Displays→ I/O Operations By Process

Displays information about database I/O operations, organized by process. Figure A–35 shows a sample I/O Operations by Process display.

```
01/25/02           I/O Operations by Process
16:19

             ---------Database -------     ---- BI -----     ---- AI -----
Usr   Name   Access     Read     Write     Read    Write     Read    Write

  0   rgw        22        8         3       13        1        2        1
  1   rgw         3        0         0        0      127        0        0
  2   rgw         3        0         0        0        0        0        0
  3   rgw      3955       49         1        0        2        0        0
  4   rgw         3        0         0        0        0        0        0
```

Figure A–35: PROMON I/O Operations By Process Display

This display shows the following information:

- **Usr** — The user number of the process.
- **Name** — The user name of the process.
- **Database Access** — The number of database access operations performed by the process.
- **Database Read** — The number of database read operations performed by the process.
- **Database Write** — The number of database write operations performed by the process.
- **BI Read** — The number of BI read operations performed by the process.
- **BI Write** — The number of BI write operations performed by the process.
- **AI Read** — The number of AI read operations performed by the process.
- **AI Write** — The number of AI write operations performed by the process.

Other Displays→ Lock Requests By User

Displays information about lock requests, organized by user. Figure A–36 shows a sample Lock Requests by User display.

```
01/25/02        Lock Requests by User
16:19

Usr   User   --- Record ---   ---- Trans ---    ---- Schema ---
Num   Name   Locks    Waits   Locks    Waits    Locks     Waits

  0   rgw        0        0       0        0        0         0
  1   rgw        0        0       0        0        0         0
  2   rgw        0        0       0        0        0         0
  3   rgw     3625        0       0        0        0         0
  4   rgw        0        0       0        0        0         0
```

Figure A–36: PROMON Lock Requests By User Display

This display shows the following information:

- **Usr Num** — The user number.
- **User Name** — The user name.
- **Record Locks** — The number of records locks acquired by the user.
- **Record Waits** — The number of times the user had to wait for a record lock.
- **Trans Locks** — Not currently implemented.
- **Trans Waits** — The number of times the user had to wait for a record lock.
- **Schema Locks** — The number of schema locks acquired by the user.
- **Schema Waits** — The number of times the user had to wait for a schema lock.

Other Displays→ Checkpoints

Displays information about checkpoints. Figure A–37 shows a sample Checkpoints display.

```
01/25/02        Checkpoints
16:19:34

Ckpt                              ------   Database Writes    ------
No.   Time      Len   Dirty      CPT Q    Scan      APW Q    Flushes

  0   10:01:54 0          0         0        0         0          0
```

Figure A–37: PROMON Checkpoints Display

This display shows the following information:

- **Ckpt. No.** — Progress assigns a number to each checkpoint sequentially.
- **Time** — The time the checkpoint began.
- **Len** — The length of time required to complete the checkpoint.
- **Dirty** — The number of modified blocks scheduled to be written.
- **Database Writes CPT Q** — The number of blocks written from the checkpoint queue by the APWs.
- **Database Writes Scan** — The number of blocks written by the APWs during the scan cycle.
- **Database Writes APW Q** — The number of blocks written by the APW queue and replaced on the least recently used (LRU) chain by APWs.
- **Database Writes Flushes** — The total number of blocks not written during the checkpoint that had to be written all at once at the end of the checkpoint.

Administrative Functions→ Resolve Limbo Transactions

Lets you resolve a limbo transaction by either committing or aborting the transaction. When you choose this option, PROMON displays the menu shown in Figure A–38.

```
1 Abort a Limbo Transaction
2 Commit a Limbo Transaction
Q Quit

Enter choice>
```

Figure A–38: PROMON Resolve Limbo Transactions Menu

Choose one of the following options:

- **Abort a Limbo Transaction** — When you choose Abort a Limbo Transaction, Progress prompts you to enter the user number of the transaction you want to abort. PROMON displays the user's number in the User column of the TC option screen. Type the user number, then press **RETURN**.
- **Commit a Limbo Transaction** — When you choose Commit a Limbo Transaction, PROMON prompts you to enter the user number of the transaction you want to commit. (PROMON displays the user number in the Usr column of the Transaction Control option screen.) Type the user number, then press **RETURN**. PROMON displays a message similar to the following:

```
User 1: commit transaction and disconnect.
```

NOTE: To commit transactions on a database that is shut down, you must use the 2PHASE RECOVER qualifier of PROUTIL.

Administrative Functions→ Adjust Latch Options

Adjusts the Spin Lock mechanism used to control access to shared system resources. Figure A–39 shows a sample Adjust Latch Options menu.

```
01/25/02        Progress Monitor (R&D)
16:34:27        Adjust Latch Options

                1. Spins before timeout        2
```

Figure A–39: PROMON Adjust Latch Options Menu

This menu contains one option:

- **Spins before timeout** — Specify a value to dynamically tune the Spin Lock mechanism. This mechanism can also be tuned using the Spin Lock Tries (`-spin`) parameter. For more information on spin locks, see the "Unproductive CPU Processing" section in Chapter 14, "Managing Performance."

Administrative Functions→ Adjust Page Writer Options

Lets you manually override APW behavior. (This should rarely be necessary. Do not change these options unless directed to do so by Progress Software Technical Support.)

Figure A–40 shows a sample Adjust Page Writer Options menu.

```
01/25/02        Progress Monitor (R&D)
16:19:34        Adjust Page Writer Options

               1. APW queue check time:        100 milliseconds
               2. APW buffers scan time:       1 seconds
               3. APW buffers per scan:        1
               4. APW writes per scan:         25
```

Figure A–40: PROMON Adjust Page Writer Options Menu

Choose one of the following options:

- **APW queue check time** — Lets you dynamically adjust the value of the Page Writer Queue Delay (`-pwqdelay`) parameter.

  NOTE: Though you can set the value of `-pwqdelay`, it is a self-tuning parameter. When the database encounters heavy loads, `-pwqdelay` decreases its value so that the APW writes more often. When the demand on the database lessens, `-pwqdelay` increases its value so that the APW writes less often. The default value, in milliseconds, for `-pwqdelay` is 100.

- **APW buffers scan time** — Lets you dynamically adjust the value of the Page Writer Scan Delay (`-pwsdelay`) parameter. The default value, in seconds, for `-pwsdelay is 1.`

- **APW buffers per scan** — Lets you dynamically adjust the value of the Page Writer Scan (`-pwscan`) parameter.

- **APW writes per scan** — Lets you dynamically adjust the value of the Page Writer Maximum Buffers (`-pwwmax`) parameter.

Administrative Functions→ Adjust Monitor Options

Displays parameters controlling the PROMON display and allows you to change them. Figure A–41 shows the Adjust Monitor Options menu.

```
01/25/02        Progress Monitor (R&D)
16:19:34        Adjust Monitor Options

               1. Display page length:            24 lines
               2. Clear screen for first page:    Yes
               3. Monitor sampling interval:      10 seconds
               4. Pause between displays:         10 seconds
               5. Pause between screens:          5 seconds
               6. Number of auto repeats:         10
```

Figure A–41: PROMON Adjust Monitor Options Menu

Adjust Monitor Options displays the following information:

- **Display page length** — The number of lines on the terminal screen.
- **Clear screen for first page** — If Yes, the system clears the screen before the main menu and before every list.
- **Monitor sampling interval** — The interval at which PROMON samples data, in seconds.
- **Pause between displays** — The delay between pages of output when in continuous monitoring mode, in seconds. (Type **u** for the Continue uninterrupted option.)
- **Pause between screens** — The delay after the last page of output when in continuous monitoring mode, in seconds.
- **Number of auto repeats** — The number of times the display is repeated when auto repeat is selected on the activity displays.

B

Setting Up Progress To Utilize Compaq's NT Clusters

This appendix describes the steps necessary to set up and configure your Compaq NT Clusters system to run Progress Database Servers and allow failover of those servers to provide continuous client access to your database servers. This appendix also includes examples used to test the functionality. Some examples include scripts to start and shut down ProService. Other examples start and stop specific database servers. There is also some sample code for the client to consider if one of the cluster nodes fails.

This appendix contains the following sections:

- Defining NT Clusters
- Configuring the NT Cluster System
- Progress Database Server Considerations and Configuration
- 4GL Client Considerations

B.1 Defining NT Clusters

An NT Cluster is comprised of two identical Windows NT Server systems connected to a shared SCSI storage device. The cluster shared disk storage can be either disks on the same SCSI bus or disks connected through a smart SCSI controller. The Compaq NT Clusters' documentation describes all the possible and supported configurations.

NT Clusters allow server application providers to create and configure their server applications to run seamlessly by providing *failover*. Failover is defined as the moving of resources from a primary node to a secondary node when the primary node either crashes or there is manual intervention to force resource reallocation from one node to another.

Within the scope of the NT Cluster, there are a few tools to manage the cluster. The primary tool is the Cluster Administrator from which the Failover Manager manages the cluster failover resources.

B.1.1 Cluster Administrator

The Cluster Administrator is a GUI tool that allows you to define Failover Groups and Failover Objects that will failover from one node to the other. Failover Groups are user-defined entities of Failover Objects. Failover Objects are the specific parts that are determined necessary to allow server products to run.

Examples of Failover Objects include disks, network protocols, and scripts. Application server providers can also provide Dynamic Link Libraries (DLLs) to make more direct calls to their products from the Failover Manager. Examples of these are the MS SQL Server, Oracle Database Server, Lotus Notes, and Web Servers.

B.1.2 Failover Manager

The Failover Manager is an image running on each node in the cluster. It handles the reading of the Failover Group information and performs the startup and shut down of the Failover Objects in the specific order in which they were defined.

The Failover Manager determines when one node has either crashed or has shutdown in some manner, then it performs actions based on the objects that were running on the failed node. The Failover Manager also allows for manual intervention to failover specific Failover Groups as if the primary node failed.

Failover Groups

Failover Groups are created by the cluster system manager using the Cluster Administrator interface. The Failover Group must contain at least one Failover Object and a cluster node must be chosen as the primary node. Within each Failover Group, the Cluster Administrator requires that there be at least one shared disk Failover Object.

After these minimum requirements are met, any number of other Failover Objects can be placed into or removed from the Failover Group. When removing Failover Objects from a Failover Group the Cluster Administrator will ensure that there is at least one object in the group.

Failover Groups can be deleted, in which case the Failover Objects that were in the group are returned to the general pool of Failover Objects. A Failover Group contains all the dependencies to allow the group to function as a unit on a node.

Failover Objects

Failover Objects are defined by the cluster manager using the Cluster Administrator interface. Failover Objects are created and placed into a pool of available objects. When the Failover Group is created, an object is placed into the group and removed from the pool of objects.

A Failover Object cannot be placed into two Failover Groups; in this scenario, two identical objects would need to be created. Failover Objects have Startup and Shutdown attributes. When a Failover Group is brought online the startup option is run; conversely, when the Failover Manager determines that the group must be failed over, the shutdown option is run.

The two primary Failover Objects are Shared Disks and TCP/IP Alias. These objects give the cluster the ability to make the cluster nodes transparent to the clients.

The shared disk objects are created automatically when the shared disks are given a name by the Cluster Administrator interface. When a disk object is placed into a Failover Group, it is brought online to the primary node by the Failover Manager. Due to the nature of SCSI, only one node can directly access or own a shared disk at any one time.

The TCP/IP alias objects are defined by the cluster manager using IP addresses that are not currently assigned to any other node. The cluster TCP/IP alias allows clients to access the nodes in the cluster without the clients needing to know to which nodes they are connected.

A TCP/IP alias object is associated with only one node at a time and migrates from the primary to the secondary node, when the Failover Manager fails over a Failover Group. In this scenario, the client just reconnects to the same name, but now the client is attached to a different node.

There can be more than one TCP/IP alias name per cluster. This happens when each node is acting as a server to a different database and defines only one IP address with the database server. Management of the TCP/IP alias name dependencies is left up to the cluster system manager. It is recommended that a TCP/IP alias object be present for each Failover Group that has objects requiring TCP/IP services.

For Progress, the only other Failover Object used is the script object. This object allows the Failover Manager to run either a script or a specific image as defined by the cluster system manager.

Scripts or images that are to be run can be placed either onto a shared storage device or onto the system disk of each node. When using the system disk, the drive letter for the system disk must be the same on both nodes or an environment variable must be used on both nodes to mask the drive letter.

When using shared storage for a script object script or image, the shared disk object must be placed before the script object in the order of the Failover Group. This ensures the disk is online before the Failover Manager attempts to run the script object.

Progress Example

For Progress, a Failover Group might be comprised of a shared disk object, a TCP/IP alias object, and a script object that would start a database server. The database server must first have a disk and then a TCP/IP address before the server can start. The script object can either start the database server directly or run a written script that starts the database server.

B.2 Configuring the NT Cluster System

The configuration and setup of the actual NT Cluster is described in the NT Clusters documentation. The order in which you install the NT Cluster software and your Progress software is very important.

If you install Progress after installing your NT Cluster software, must reboot the node to allow the NT Cluster software to read the Registry and system environment variables so it knows about the Progress software.

If you install Progress first, then install the NT Cluster software, there are no such problems. Each node in the cluster must have Progress installed onto its system disk. You should install identical versions and service packs of Progress on both nodes to ensure the same operating environment.

One other configuration issue that you should be aware of is that the system disk drive letter must be the same for both nodes in your cluster. (Usually this letter is either C: or D: depending on how you partition your node's system disk.) Using the same system disk drive letter lets you create and run scripts from your NT Cluster Failover Manager. The script location is stored in the Failover Manager and must be listed with the same physical location on both systems.

After the two systems have the NT Cluster software running, use the Windows NT Disk Administrator to assign a drive letter to each shared disk and then use the Cluster Administrator to assign a name to your disks and to create your disk Failover Objects.

Use the Cluster Administrator to create your TCP/IP alias Failover Objects also. Once they are created, these objects will appear in the Cluster Administrator display and you can begin to create your Failover Groups for Progress.

B.3 Progress Database Server Considerations and Configuration

On the server side, think of the cluster as one node. Although from a management viewpoint, you will still have nodes onto which you must install, configure, and update the product.

The original setup is the most time consuming because you must install, configure, and maintain the Progress environment in addition to updating and maintaining the TCP/IP `services` file on each node. However, from a user's or an application's viewpoint there is only one host and one service.

For TCP/IP, the service name you associate with your database server is a name that will be used cluster-wide. Therefore, you must edit the TCP/IP `services` file on both nodes and also add the same service name and socket number on each node. You must use the same names and numbers, because in a failover, the secondary node starts the database server and your clients must reconnect to your service by node and service number.

After Progress is installed on both nodes, you must determine which cluster shared disks will hold your database. Next, use the Cluster Administrator to create a new Progress Failover Group. Place the stored disk objects into the group. Doing this places the disks online on the node that you have chosen to be the primary node. From the primary node, use the Progress Database Administration tools to create or copy a database onto the shared disks.

Next, use ProControl to configure the database server information, using a cluster TCP/IP alias for the Host (`-H`) parameter name and the "cluster-wide" service name from the `services` file for the Service (`-S`) parameter name.

You must duplicate this effort on both nodes, so you should have a ProControl screen visible on both nodes so you can be sure the duplication is exact.

Within ProControl you can choose to have the database server started automatically or not. If you have only one cluster shared disk, you should allow ProControl to start the database servers automatically when ProControl starts ProService.

However, if you have multiple cluster shared disks and you have chosen to have each node act as the primary node for specific database servers, this approach is not recommended. At this time, ProControl only knows three commands, Start, Stop, and Status. Therefore, if you have database servers running on both nodes and if one node fails and those database servers failover to the secondary node, you would have to Stop and Start ProControl in order to start the automatic database server startup. Doing this disrupts current users of the secondary node.

In the single cluster shared disk model, ProControl should not be already running on the nonprimary node. So, in a failover, you start ProControl with automatic database server startup set and your database servers will be started for you.

Continue creating Failover Groups and adding/modifying ProControl for each database server on each shared disk.

After this is done, use the Cluster Administrator to create Script Failover Objects to be placed in your Progress Failover Groups. These script objects can be either actual scripts or the direct command-line interface into ProControl. In either case, you must use the ProControl command-line interface. The following sections describe the command-line interface and give you some examples for scripts.

B.3.1 ProControl Command-line Interface

In order to start Progress database servers on an NT node, ProControl simply runs the PCCmd command. This image controls either starting or stopping ProService and the various database servers. For the entire syntax of the PCCmd command, see the *Progress Startup Command and Parameter Reference*.

Within your scripts or script objects, use the following syntax and parameters based on your requirements:

SYNTAX

```
%DLC%\BIN\pccmd.exe P1 P2 [P3]
```

%DLC% is a system-wide environment variable defined during Progress installation. The parameters to `ppcmd.exe` dictate what is done. The following is the list of parameters you will need:

P1 - Parameter 1

Use P1 to specify one of the following values for the type parameter:

- PROSERVICE for ProService options
- DATABASE for Database options

P2 - Parameter 2

Use P2 to specify one of the following values for the command parameter.

START to start either ProService or the Database Server named in P3.

STOP to stop either ProService or the Database Server named in P3.

STATUS to return one of the following:

- Current status of ProService or the Database Server in P3
- Use %ERRORLEVEL% to detect return status in script
- A return of 0 indicates request is running (started)
- A return of 1 indicates request is not running (stopped)
- Else, check if %DLC% is defined correctly.

[P3] - Parameter 3

Use [P3] to specify a value for the identifier parameter. [P3] is required if Parameter 1 is DATABASE.

If a wrong parameter is passed to PCCmd, it will be signaled to the standard output and no action will be taken.

EXAMPLES

Some command-line examples include:

```
%DLC%\bin\pccmd.exe ProService Status
```

A return of 0 from this call means ProService is already running:

```
%DLC%\bin\pccmd.exe ProService Start
```

If run while already running, a message displays stating that ProService is already running; otherwise, a successful run will return 0:

```
%DLC%\bin\pccmd.exe ProService Stop
```

If run while already stopped, a message displays stating that ProService is already stopped; otherwise, a successful run will return 0:

```
%DLC%\bin\pccmd.exe Database Status DEMO
```

A return of 0 from this call indicates the Database server for DEMO is already running:

```
%DLC%\bin\pccmd.exe Database Start DEMO
```

A return of 0 from this indicates the Database server for DEMO was started successfully. If a start is attempted twice, 0 will still be returned and no message is displayed:

```
%DLC%\bin\pccmd.exe Database Stop DEMO
```

A return of 0 from this indicates the Database server for DEMO was stopped successfully. If a stop is attempted twice or attempted upon a stopped database server, a 1 will be returned and no message is displayed.

Using the above commands, you can create either an entire script to be run or Failover Objects for ProControl and each database server to be placed into a Progress Failover Group.

Determining whether or not to create a script depends on the environment. In the simple case of one database server per cluster, by simply starting and stopping ProService with the database server being automatically started and shutdown by ProService, a script is unnecessary. In this case, using the direct command-line start/stop ProService as the startup and shutdown options to the Failover Object should be used. In the more complicated case of multiple database servers on both nodes, you should use scripts. The scripts can then use the “status” option to determine whether or not ProService is running, as well as determine whether or not a database server is running for the particular object.

NOTE: The Failover Manager is documented to perform startup in the order of the Failover Groups and shutdown in the reverse order of the Failover Groups.

Startup of objects within each group is performed in the order listed within the group, while shutdown is executed in reverse order. Therefore, the cluster system manager must properly configure the order so that each Progress database server will have the necessary disk and network objects in place before attempting to start. This becomes especially tricky when dealing with multiple TCP/IP alias objects. The script objects to start and stop the database should be placed into the same group as the TCP/IP alias name for each database server that uses a particular TCP/IP alias name.

ProService Startup/Shutdown Script Examples

The following are examples of scripts that can be used to start and stop ProService. The scripts are stored on a shared disk, which is assigned the drive letter S. The Failover Object is configured to run the scripts for start and stop, respectively:

StartProService.cmd

```
StartProService.cmd
ECHO OFF
REM This is a Cluster Startup Script for the Cluster Failover Manager to
REM execute when the cluster has a problem. Its primary purpose is to restart
REM ProService if it isn't already started on the node.
SET LOG=S:\ProgressScripts\StartProService.log
ECHO "Running on %COMPUTERNAME%" >> %LOG%
DATE /T >> %LOG%
%DLC\bin\PCCMD.EXE ProService Status >> %LOG%
IF %ERRORLEVEL% == 1 GOTO notstarted
  ECHO "ProService already running >> %LOG%
  GOTO done

:notstarted
ECHO "Starting ProService"  >> %LOG%
%DLC%\bin\PCCMD.EXE ProService Start >> %LOG%
ECHO "Return value from start is %ERRORLEVEL%" >> %LOG%

:done
```

StopProService.cmd

```
ECHO OFF
REM This is a Script for the Cluster Failover Manager to execute when the
REM cluster has a problem. Its primary purpose is to Stop ProService if
REM it isn't already stopped on the node.

SET LOG=S:\ProgressScripts\StopProService.log

ECHO "Running on %COMPUTERNAME%" >> %LOG%
DATE /T >> %LOG%

%DLC\bin\PCCMD.EXE ProService Status >> %LOG%
IF %ERRORLEVEL% == 1 GOTO notstarted
  ECHO "Shutting down ProService" >> %LOG%
  %DLC%\bin\PCCMD.EXE ProService Stop >> %LOG%
  ECHO "Return value from shutdown is %ERRORLEVEL%" >> %LOG%
  GOTO done

:notstarted
ECHO "ProService is already shutdown >> %LOG%

:done
```

Specific Database Startup/Shutdown Script Examples

Within ProService startup or shutdown scripts, or within a separate script, once ProService has started, then specific database servers can be started. Conversely, before ProService is shut down, the database servers should be shut down as well.

If you have multiple databases, you might want to create a generic script that can be run with an input parameter for the database name.

The following examples assume that you have set up ProControl to have a database server identified as CluDemo1. (Note it is very important to match the case between the script and ProControl.) If the scripts are contained within the ProService startup/shutdown scripts (either directly or through a reference call), then only one object needs to be added to the Failover Group. However, if they are separate scripts, then multiple objects are needed for each database to be started/shut down by the Failover Group.

The following are code fragments to Start and then Stop the Progress database server for database CluDemo1. The code assumes the same as the Start and Stop ProService examples above with the cluster shared disk as drive letter S:

StartCluDemo1.cmd

```
ECHO OFF
REM This is a Script for the Cluster Failover Manager to execute when the
REM cluster has a problem.
REM It's primary purpose is to Start the Database server for CluDemo1.

SET LOG=S:\ProgressScripts\StartCluDemo1.log
ECHO "Running on %COMPUTERNAME%" >> %LOG%
DATE /T >> %LOG%

%DLC%\bin\pccmd.exe Database Status CluDemo1 >> %LOG%
IF %ERRORLEVEL% == 1 GOTO startcludemo1
    ECHO "Server for CluDemo1 already started" >> %LOG%
    GOTO done cludemo1start

:startcludemo1
    ECHO "Starting database server for CluDemo1" >> %LOG%
    %DLC%\bin\PCCMD.EXE Database Start CluDemo1
    ECHO "Return value from startup is %ERRORLEVEL%" >> %LOG%

:donecludemo1start
```

B.4 4GL Client Considerations

On the client side, you must consider a few things. First, the client no longer connects to a specific host using `-H`. Instead, the client connects to an NT Cluster TCP/IP Alias. Therefore, any code in which you specify the host to connect to, you must modify the code to use the cluster alias in use for the database server. You can still connect directly to a host; however, if that host is down or not serving the database, then your client will not reconnect unless you change the value of `-H` to the name of the other node.

Secondly, if you do not already have in your client P-Code error checking for the STOP signal, you must add code to handle the STOP signal. See the section on "How Progress Handles System and Software Failure" section in the *Progress Programming Handbook* for details on the STOP condition handling.

The following code examples exhibit the types of error handling that need to be done to support the cluster failover from the client side. When a node fails and the Cluster Manager migrates the database server from one node to another, the Progress client is given a STOP signal, which is trapped and handled in the code.

The client automatically receives a couple of Progress-generated error dialog boxes indicating the last write was not performed and potentially the last transaction was not completed. Alert your client users to these messages, so they know that they must re-enter their last transaction.

The following examples assume that you have a Database server running on your Windows NT cluster with a Service (-S) name of ‘cludemo1’ using the Host (-H) name ‘cluster’. The demonstration is strictly to exhibit what type of programming must be done to make this work. It is not a full-featured application. The CluDemo1 database is a copy of the Progress SPORTS demonstration database with no other changes.

Lastly, one other programming consideration that is exhibited by the demonstration program: Do you bring your client back to the point where the failure occurred, or do you start your client at the beginning? In other words, how much state do you keep within your code? This example tries to bring the client back to the last customer record being updated before the STOP was raised. Therefore, the state that is kept within this code is the last customer record chosen:

repeat.p

```
DEFINE NEW GLOBAL SHARED VARIABLE cur-cust AS INTEGER INITIAL 0.

REPEAT ON ERROR UNDO, LEAVE
  ON STOP UNDO, RETRY:
    IF RETRY
    THEN DO:
        MESSAGE "The STOP condition has occurred." SKIP
          "Do you wish to continue?" VIEW-AS ALERT-BOX QUESTION
          BUTTONS yes-no UPDATE continue-ok AS LOGICAL.
        IF NOT continue-ok
        THEN UNDO, LEAVE.
    END.

   IF NOT CONNECTED ("cludemo1") THEN
       connect cludemo1 -S cludemo1 -H cluster -N tcp.

   IF NOT CONNECTED ("cludemo1")
    THEN DO:
        RUN updcust.p.
        MESSAGE "Exit from updcust.p" SKIP
         "Do you wish to continue?" VIEW-AS ALERT-BOX QUESTION
          BUTTONS yes-no UPDATE continue-ok.
        IF NOT continue-ok
        THEN LEAVE.
        ELSE cur-cust = 0. /* Reinitialize */
    END.
    ELSE
    DO:
      MESSAGE "Database not available." SKIP
       "Do you wish to continue?" VIEW-AS ALERT-BOX QUESTION
        BUTTONS yes-no UPDATE continue-ok.
   IF NOT continue-0k
   THEN LEAVE.
   END.
end.
```

updcust.p

```
/* If never run before cur-cust is 0 */
DEFINE SHARED VARIABLE cur-cust AS INTEGER.

DEFINE VARIABLE first-time AS LOGICAL INITIAL TRUE.

REPEAT WITH 1 COLUMN 1 DOWN:
    IF first-time AND cur-cust NE 0
    THEN DO:
        FIND customer WHERE customer.cust-num = cur-cust NO-ERROR NO-WAIT.
    END.
    ELSE
    DO:
        PROMPT-FOR customer.cust-num.
        FIND customer USING cust-num NO-ERROR NO-WAIT.
    END.

    first-time = FALSE.

    IF NOT AVAILABLE customer
    THEN DO:
        MESSAGE "Customer record " INPUT customer.cust-num " not found."
SKIP
                "Do you wish to continue?" VIEW-AS ALERT-BOX QUESTION
                BUTTONS yes-no UPDATE continue-ok AS LOGICAL.
        IF NOT continue-ok
       THEN LEAVE.
       cur-cust = 0.
    END.
    ELSE
    DO:
        cur-cust = customer.cust-num.
        DISPLAY customer.cust-num.
        UPDATE name address city state postal-code credit-limit.
    END.
END.
```

Index

A

-a startup parameter, RFUTIL utility 19–146, 19–150, 19–153

Abnormal termination 8–21

Abort a Limbo Transaction option
  PROMON utility 19–48, A–68

Absolute-path database 1–15, 4–13, 4–23, 9–6

Absolute-path, *See*Database

Activity option
  PROMON utility 19–37

ADD qualifier
  PROSTRCT utility 19–56

Addressing
  Progress brokers/servers 5–7
  Progress hosts 5–7

-adminport startup parameter 18–8

AdminServer 5–2
  on UNIX 19–8

AdminServer Port (-adminport) startup parameter 18–8

After-image Buffers (-aibufs) startup parameter 14–28, 17–4, 18–9

After-image extents 8–4
  backing up 6–2, 8–19, 8–20
  backups 6–5
  choosing backup media 11–9
  defined 11–2
  displaying information 19–144
  emptying 19–142
  fixed-length 11–3
  on-line backups 6–5, 19–13
  variable length 11–3
  variable-length 11–3

After-image file
  scanning 19–146
  truncating 19–145, 19–147

After-image Filename (-a) startup parameter
  RFUTIL utility 19–146, 19–150, 19–153

After-image Filenmae (-a) startup parameter 11–12

After-image files. *See* After-image extents

After-image Stall (-aistall) startup parameter 11–7, 18–10

After-image writer 17–4

After-image writer process
  stopping 17–4

After-image writers (AIW) 14–28
PROAIW utility 17–4
starting 5–11
stopping 5–11

After-imaging
block size 14–29
buffers 14–28
changing block size 19–147
described 8–4, 11–5
disabling 19–141
enabling 11–5, 19–140
I/O impact 14–27
log-based replication 15–7
monitoring activity 14–27
roll forward recovery 6–13
two-phase commit 12–6

-aiblocksize startup parameter
RFUTIL utility 19–147

-aibufs startup parameter 18–9

AIMAGE AIOFF qualifier
RFUTIL utility 19–139

AIMAGE BEGIN qualifier
RFUTIL utility 19–140

AIMAGE END qualifier
RFUTIL utility 19–141

AIMAGE EXTENT EMPTY qualifier
RFUTIL utility 19–142

AIMAGE EXTENT FULL qualifier
RFUTIL utility 19–143

AIMAGE EXTENT LIST qualifier
RFUTIL utility 19–144

AIMAGE NEW qualifier
RFUTIL utility 19–145

AIMAGE SCAN qualifier
RFUTIL utility 19–146

AIMAGE TRUNCATE qualifier
RFUTIL utility 19–147

-aistall startup parameter 11–7, 18–10

AIW. *See* After-image writer

Application data area 1–8

Application data areas
obtaining descriptions 7–28

Application security
connection 10–5
designating a security administrator 10–5
passwords 10–2
user list 10–2

Architecture 1–2 to 1–3, 1–9

Archiving
backups 7–8

Areas
application data 1–8
control 1–8
primary recovery 1–8
schema 1–8
storage 1–7
transaction log 1–8

AS/400 Host Name (-H) startup parameter 18–24

Asynchronous page writers (APWs) 14–14
optimal number 5–10, 14–14
scanning cycles 14–15
starting 5–10, 14–17
stopping 5–10, 17–5

Asynchronous replication 15–3

Audience 2–xxiii

Authorization keys
inserting 19–122
setting for a database 19–90

Auto Server (-m1) startup parameter 18–27

Auto Server server type 18–27

Auto-connect
dumping records 13–9

AutoConvert 4–24

B

-B startup parameter 18–10
  RFUTIL utility 19–151, 19–153

-b startup parameter
  PROSHUT command 5–12, 17–11

Backup blocks
  cyclic redundancy check 7–24

Backups
  after-imaging 6–8
  archiving 7–8
  before-image files 6–2
  developing strategy 6–9
  displaying date and time of last 19–64
  important files 6–2
  marking databases 19–149
  media 6–8, 7–10
  options 7–5
  schedules 8–4
  structure files 6–2
  testing 7–6
  types 6–3
    full 7–9
    incremental 6–4
    non-Progress 7–13
    off-line 6–6, 7–9
    on-line 6–5, 7–10
  unscheduled 6–9

Bad load
  reconstructing records 13–30

-baseindex startup parameter 18–11

-basetable startup parameter 18–12

Before-image
  cluster size 19–77

Before-image block size 14–23, 19–132

Before-image buffers 14–20

Before-image Buffers (-bibufs) startup parameter 14–20, 17–6, 18–13

Before-image Cluster Age (-G) startup parameter 18–23

Before-image clusters 14–17
  number 14–22
  size 14–21, 19–132

Before-image extents. *See* Before-image files

Before-image files 8–3
  backing up 6–2
  truncating 8–24, 19–44, 19–132
  what to do when full 8–22
  where to store 2–3

Before-image writers 14–20, 17–6
  starting 5–11, 18–13
  stopping 5–11, 17–6

Before-imaging
  described 8–3
  I/O impact 14–18
  -Mf startup parameter 18–30
  monitoring activity 14–18
  suppressing BI writes 14–24

-bi startup parameter
  PROUTIL TRUNCATE BI utilitye 19–132

BI Threshold
  adjusting 14–26
  setting 14–25
  stalling 14–26

-biblocksize startup parameter PRUTIL TRUNCATE BI utility 19–132

BIGROW qualifier
  PROUTIL utility 14–23, 19–77

binary dump 13–10

binary load 13–21, 19–118

-bistall startup parameter 18–13

-bithold startup parameter 14–25, 18–14

bithreshold parameter 17–7

BIW. *See* Before-image writers

BKTBL chain. *See* Block table chain

Block Access option
  PROMON utility 19–32

Block sizes
  after-image 19–147
  before-image 19–70, 19–132
  displaying 19–64

Block sizes. *See also* Structure description file

Block table chain 14–15

Blocking Factor (-bf) startup parameter
  PROBKUP utility 7–3, 7–25, 19–11

Blocks
  after-image 14–29, 19–147
  before-image 14–23, 19–132
  database
    displaying information 19–64
    number used 19–44
  empty 19–44
  error correction 7–25
  free chain 19–81, 19–89
  index 19–89, 19–103
  recovering corrupted 7–24
  RM chain 19–81

Blocks in Database Buffers (-B) startup parameter 18–10

-bn startup parameter 5–13, 17–12

Bold typeface
  as typographical convention 2–xxvi

BPRO utility
  no-integrity mode 13–36

Brokers
  addressing 5–7
  multi-broker access 5–8
  shutting down 5–12
    permissions 17–14
  starting 5–7
    servers for remote users 5–5

Buffer hits
  optimal number 14–11

Buffer Size (-B) startup parameter 14–11

Buffered I/O (-r) startup parameter 18–37
  RFUTIL utility 19–151, 19–153

Buffers
  after-image 14–28
  before-image 14–20
  database. *See* Database buffers
  server startup parameters 18–2

Buffers (-B) startup parameter
  RFUTIL utility 19–151, 19–153

BUILDDB qualifier 8–25

Bulk loader description files 13–28, 19–78
  creating 13–25
  modifying 13–26

Bulk Loader utility 13–27
  loading database contents 13–27

BULKLOAD qualifier
  PROUTIL utility 13–27, 19–78

BUSY qualifier
  PROUTIL utility 7–9, 7–13, 19–79

-by startup parameter
  PROSHUT command 5–13, 17–12

C

-C disconnect usernum qualifier
  PROSHUT command 17–12

-C list qualifier
  PROSHUT command 17–12

Case Code Page (-cpcase) startup parameter 18–15

Chains
  empty 14–20
  filled 14–20

CHANALYS qualifier
PROUTIL utility 19–81

Changing your password 10–9

Character mapping 13–14

Character sets
displaying 19–64
IBM850 19–84

Characters
allowed for database names 3–8

Checking indexes 19–107

Checkpointing 14–14, 14–17, 14–21

Checkpoints A–67

Client/server Progress
addressing 5–7
broker/server addressing 5–7

Clients
multiple 1–9

Cluster size
before-image 19–132
specifying 19–77

Clusters
before-image 14–21, 19–132

CMPDB Qualifier 19–7

Code pages
converting 19–82

CODEPAGE-COMPILER qualifier
PROUTIL utility 19–82

Collation Code Page (-cpcoll) startup parameter 18–16

Collation names
displaying 19–64

-com startup parameter
PROBKUP utility 19–12

Commit a Limbo Transaction option
PROMON utility 19–48, A–68

Compacting indexes 9–11, 14–41, 19–109

Compiling
word rules files 19–136
work-break table 19–135

Compression (-com) startup parameter
PROBKUP utility 19–12

Configuration utilities
DBMAN 5–3, 5–4, 19–3
PROADSV 5–3, 19–8
SQLEXP 5–3

Configurations 1–9
database location 1–10
distributed database 1–11
federated database 1–11
number of databases 1–10
system platforms 1–9

conmgr.properties file 5–3, 17–9, 19–3, 19–5

Connection Modes 1–9
batch 1–10
interactive 1–10
multi-user 1–10
single-user 1–9

Connection security
establishing 10–5

Consolidating disk space 14–37

Control area 1–8

CONV89 4–18

CONV89 Qualifier 19–83

CONV89 qualifier
PROUTIL utility 4–15

CONVCHAR qualifier
PROUTIL utility 19–84

Conversion Map (-convmap) startup parameter 18–14, 18–15

Conversion map files 19–82

Converting
  database character sets 19–84
  text file character sets 19–87
  Version 8 databases to Version 9 4–15, 4–18

Converting codepages 19–82

CONVFILE qualifier
  PROUTIL utility 19–87

-convmap startup parameter 18–14, 18–15

Coordinator (-crd) startup parameter
  PROUTIL utility 12–7

Coordinator databases 12–3, 19–71
  assigning 12–7
  nickname 12–7

Coordinator Information option
  PROMON utility 19–49

Copying a database 4–22

-cpcase startup parameter 18–15

-cpcoll startup parameter 18–16

-cpinternal startup parameter 18–17

-cplog startup parameter 18–18

-cpprint startup parameter 18–18

-cprcodein startup parameter 18–19

-cpstream startup parameter 18–20

-cpterm startup parameter 18–21

-crd startup parameter
  PROUTIL 2PHASE BEGIN utility 19–71
  PROUTIL 2PHASE MODIFY utility 19–75

CREATE qualifier
  PROSTRCT utility 11–5, 19–58

Creating
  a void database 4–11
  bulk loader description file 13–25
  database 4–2
  databases 4–1, 4–10, 4–14
    new Progress 4–1
    PRODB utility 4–12
    starting version 13–30
  incremental data definition files 13–7
  storage areas 4–8
  structure description file 4–3

Cyclic Redundancy Check (CRC)
  codes
    backup 7–24

D

Data
  bulk loading 13–27, 13–36, 18–25
  dumping 13–14
  loading 13–22

Data Administration tool
  adding a user 10–7
  changing a password 10–9
  creating a database 4–14
  deleting a user 10–9
  designating a security administrator 10–5
  dumping
    auto-connect records 13–9
    creating an incremental data definition file 13–7
    data definitions 13–3, 13–6
    database contents 13–14
    sequence definitions 13–8
    sequence values 13–17
    SQL view file contents 13–18
    user table contents 13–17
  freezing and unfreezing tables 10–12
  loading
    database contents 13–22
    recovering from load errors 13–30
    sequence values 13–24
    SQL view file contents 13–24
    user table contents 13–23
  running a user report 10–10

Data definition files
creating an incremental file 13–7

Data definitions
loading 13–19

Data Dictionary
adding a user 10–7
changing a password 10–9
copying a database 4–22
creating an incremental data definition file 13–7
deleting a user 10–9
designating a security administrator 10–5
dump and load limitations 13–3
dumping
auto-connect records 13–9
data definitions 13–3, 13–6
database contents 13–14
database definitions 13–3
sequence definitions 13–8
sequence values 13–17
SQL view file contents 13–18
user table contents 13–17
freezing and unfreezing tables 10–12
loading
database contents 13–22
database definitions 13–18
recovering from load errors 13–30
sequence values 13–24
SQL view file contents 13–24
user table contents 13–23
running a user report 10–10

Data extents
backing up 6–2

Data replication
database ownership models 15–3
log-based site replication 15–2
replication models 15–2
trigger-based replication 15–2

Data Types
limits 3–10

Data Values
limits 3–10

Database
absolute-path 1–15, 1–16
creating 4–2
relative-path 1–15 to 1–16

Database administrator (DBA)
designating 10–3
for Progress 4GL 10–3
for SQL-92 10–3
sysprogress 10–3

Database blocks
number used 19–44

Database buffers
buffer pool size 14–11
described 14–6
evicting 14–10
eviction process 14–10
modified 14–15
private read-only buffers 14–12
Progress usage 14–9

Database Configuration 1–10
cluster 1–14
distributed 1–11
federated 1–11
multi-tier 1–14

Database configurations
conmgr.properties file 5–3

Database contents
dumping 13–14
loading
bulk loader 13–27

Database definitions
dumping 13–3
loading 13–2, 13–18

Database files
backing up 6–2
sample locations, safe recovery 8–6

Database integrity 18–25

Database limits
block sizes 3–2
index limits 3–5
number of sequences 3–5
number of simultaneous transactions 3–7
number of users 3–7
records per block 3–5
size 3–6

Database Location 1–10
local 1–10
remote 1–10

Database names
characters not used 3–8

Database Quiet Points 7–11, 17–7

Database records
reading 14–9

Database servers
shutting down 7–13

Database sizes 19–44

Database startup parameters
alphabetical listing 18–8

Database state
empty 19–44

Database Status option
PROMON utility 19–43

Database structure
creating 4–10
maintaining 9–1

Database triggers
dumping and loading 13–2

Databases
absolute-path 1–15, 4–13, 4–23, 9–6
activity 19–37
analyzing fragmentation 14–37
analyzing index utilization 14–39
analyzing structure 14–37
authorization key 19–90, 19–122
backup media 7–10
backups 19–10
blocksize 19–58
configuring properties 19–23
converting
Version 8 to Version 9 4–15
converting character sets 19–84
converting from single to multi-volume 19–17
converting single-volume Version 8 4–15, 4–16
coordinator 12–3, 19–71
copying 4–22, 19–17
creating 4–1, 4–14, 19–19
a starting version 13–30
new Progress database 4–1
PRODB utility 4–12
deleting 4–1, 4–24, 19–22
determining if in use 7–9, 7–13, 19–79
determining user mode 19–101
disabling after-imaging 19–141
disk requirements 2–3
distributed 1–11
dumping 13–2
contents 13–14
definitions 13–3
sequences 13–8
dumping and reloading 13–32
limitations 13–3
quickly 13–36
events 19–37
federated 1–11
formulas for calculating size 2–3
fragmentation 19–124
loading 13–2, 19–78
loading contents 13–22
bulk loader 13–27
loading definitions 13–18
location 1–10
marking as backed up 19–149
marking backed up 7–14
monitoring 19–25, A–1
multi-volume
creating a void multi-volume database 11–5
PROSTRCT utility 19–54
naming conventions for different operating systems 3–8
nickname 12–7
preparing for two-phase commit 12–7

reconstructing with after-imaging file 19–150
record sizes 19–124
relative-path 4–13, 4–23, 9–6
restoring 7–25, 8–17, 8–19
roll forward recovery 8–2
running Progress in no-integrity mode 13–36
servergroup 17–9
shutting down 5–12
PROMON 5–15
shutting down the server 7–13
states A–10
statistics for buffer pool access 19–32
status report for restore procedure 7–27
storage area location 2–3
table limits 3–5
unlocking 19–66
updating an existing schema 13–19
using one table in multiple databases 13–32
verifying backups 7–23
void 4–11
word break tables 19–135
word rules files 19–136

DBANALYS qualifier
PROUTIL utility 19–89

DBAUTHKEY qualifier
PROUTIL utility 19–90

DBIPCS qualifier
PROUTIL utility 19–92

DBMAN 5–3, 5–4, 19–3

Delayed BI File Write (-Mf) startup parameter 14–25, 18–30

Deleting
a user from user list 10–9
databases 4–1, 4–24

Description files
See also Structure description file
Bulk Loader 13–28, 19–78

Direct I/O 14–30

Direct I/O (-directio) startup parameter 18–21

-directio startup parameter 18–21

Disconnect a User menu option
PROMON utility 19–45

Disconnect usernum qualifier
PROSHUT command 17–12

Disk I/O
balancing
multi-volume databases 14–6

Disk space
consolidation 14–37
estimating 2–3
formulas for calculating database size 2–3
maintenance 13–32
using efficiently 13–32, 14–37
what to do when full 8–21

Distributed
transactions 12–2

DOS
and on-line backups 19–13
backup utilities 6–4, 11–10

DUMP qualifier
PROUTIL utility 19–94

Dumping 13–2, 19–94
auto-connect lists 13–9
binary 13–10
data definitions 13–3
database contents 13–14
database sequences 13–8
databases 13–36
disk space maintenance 13–32
limitations 13–3
SQL view file contents 13–18
SQL-92 19–154
triggers 13–2
user table contents 13–17

Dumping tables 8–27

E

Emergency Shutdown menu option
PROMON utility 19–45

Empty blocks 19–44

ENABLEPDR qualifier
PROUTIL utility 19–100

Enabling
after-imaging 11–5
two-phase commit 12–7

Encoding passwords 10–2

Endtime startup parameter
RFUTIL utility 19–150, 19–152

Endtrans startup parameter
RFUTIL utility 19–151, 19–152

Error correction
PROBKUP utility 7–25, 19–12

ESQL
-pp startup parameter 18–37

ESQL PROPATH (-pp) startup parameter 18–37

-estimate startup parameter
PROBKUP utility 7–2, 19–11

Event-Level (-evtlevel) startup parameter 18–22

Events 19–37

-evtlevel startup parameter 18–22

Example procedures 2–xxxi

Extent types
for structure description file 4–9

Extents 1–7
See also After-image extents
adding 9–5, 19–56
to existing storage areas 9–7
creating 19–58
creating structure description file 19–60
defining 4–7
size 4–9
displaying information 19–144
fixed-length 1–7, 4–9
marking empty 19–142
maximum length 4–9
removing 9–8, 19–61
repairing 19–57, 19–63
variable-length 1–7, 4–9
function 4–9
maximum number 4–9

F

-F startup parameter
PROSHUT command 5–13, 17–12

f token
for data file length in structure description file 4–9

Fields
formulas for calculating storage values 2–4

File descriptors
defined 1–13, 14–37

File extensions
See listing for specific extension

File handles
maximum number 3–8

Files
after-image 8–4
backing up 6–2
before-image 8–3, 8–24, 19–132
converting character sets 19–87
fragmentation 14–37
log 19–24
moving to alternate locations 14–20
structure description 19–58, 19–60
transaction log (.tl) 12–6

Fixed-length extents 4–9
after-image 11–3
on-line backups 6–5

Fixed-length, *See*Extents

Fixing indexes 19–111

Force Access (-F) startup parameter
  PROSHUT command 5–13, 17–12
  PROUTIL utility 8–28

Fragmentation 14–37
  displaying degree 19–124

Free blocks
  monitoring for multi-volume databases 9–2

Free chain blocks 19–81, 19–89

Free space 19–44

Freezing tables
  permissions needed 10–12

Full backup restorations
  examples
    NT 7–30
    UNIX 7–29

Full backups 7–9
  examples
    NT 7–17
    UNIX 7–4

Full Verify (-vf) startup parameter
  PROREST utility 7–7, 7–26, 19–52

G

-G startup parameter 18–23
  PROUTIL utility 19–132

Group Delay (-groupdelay) startup parameter 18–23

-groupdelay startup parameter 18–23

-Gw startup parameter
  PROSHUT command 17–12

H

-H startup parameter 18–24
  PROSHUT command 5–13
  specifying 5–7
  PROSHUT command 17–12

Hardware failures
  protecting against 2–3

-hash startup parameter 18–24

Hash table
  defined 14–12

Hash Table Entries (-hash) startup parameter 14–12, 18–24

Help
  Progress messages 2–xxxiv

HOLDER qualifier
  PROUTIL utility 19–101

Host addressing 5–7

Host Name (-H) parameter
  specifying 5–7

Host Name (-H) startup parameter 18–24
  PROSHUT command 5–13, 17–12

I

-i startup parameter 18–25

I/O
  after-image 14–27
  before-image 14–18
  bottlenecks
    eliminating 14–6
  buffered 19–151, 19–153
  databases 14–6, 14–7
  performance impact 14–6

I/O operations A–65

IBM Codepage 850 character set 19–84

IDXANALYS qualifier
PROUTIL utility 19–103

IDXBUILD qualifier
PROUTIL utility 14–42, 19–104

IDXCHECK qualifier
PROUTIL utility 19–107

IDXCOMPACT qualifier
PROUTIL utility 9–11

IDXFIX qualifier
PROUTIL utility 19–111

IDXMOVE qualifier 9–11

Incremental backup restorations 7–27
examples
NT 7–31
UNIX 7–29

Incremental backups 6–4
examples
NT 7–20
UNIX 7–14
overlap factor 19–12

-incremental startup parameter
PROBKUP utility 7–2, 7–6, 19–10, 19–12

Index Base (-baseindex) startup parameter 18–11

Index blocks 19–89
displaying information 19–103

Index Range Size (-indexrangesize) startup parameter 18–26

Indexes
adding 9–9
analyzing utilization 14–39
checking 19–107
compacting 9–11, 19–109
corruption 14–42
fixing 19–111
fragmentation 14–39
moving 9–11, 19–114
reactivating 14–47
rebuilding 13–29, 14–42, 19–104
sorting
formulas to determine free space for rebuilding 14–43

INDEXMOVE Qualifier
PROUTIL utility 19–114

-indexrangesize startup parameter 18–26

INPUT FROM statement 13–16

Integrity
database 18–25, 19–52

Internal Code Page (-cpinternal) startup parameter 18–17

International character sets
specifying 5–6

-io startup parameter
PROBKUP utility 19–12

IOSTATS qualifier
PROUTIL utility 19–116

Italic typeface
as typographical convention 2–xxvi

IXANALYS qualifier
PROUTIL utility 14–39

K

Keys
authorization 19–90, 19–122

Keystrokes 2–xxvii

Kill All menu option
PROMON utility 19–45

Kill users (-by) startup parameter
PROSHUT command 5–13, 17–12

L

-L startup parameter 18–26

Large transactions
  effect on lock table 18–27

Least recently used (LRU) chain 14–16

Limbo lock 19–36

Limbo transactions
  aborting 19–76
  committing 19–76
  defined 12–5
  detecting 12–10
  how Progress processes react 12–10
  PROUTIL 2PHASE COMMIT utility 19–73
  resolving 12–11

Limits
  database
    number of simultaneous transactions 3–7
    number of users 3–7
  database names 3–8
  dumping and reloading databases 13–3
  for database names 3–8
  operating systems 3–8

LIST qualifier
  PROSTRCT utility 19–60

List qualifier
  PROSHUT command 17–12

LOAD qualifier
  PROUTIL utility 19–118

Loading 13–2, 19–118
  binary 13–21
  database contents 13–22
    bulk loader 13–27
  database definitions 13–18
  databases
    disk space maintenance 13–32
  SQL view file contents 13–24
  SQL-92 19–159
  triggers 13–2
  user table contents 13–23

Lock requests A–66

Lock Table Entries (-L) startup parameter 18–26

Locking
  resource
    UNIX 14–33

Locking and Waiting Statistics option
  PROMON utility 19–27, 19–30

Locks
  displaying contents of record locking table 19–35
  limbo 19–36
  purged 19–36
  queued requests 19–36
  upgrade requests 19–36

Log (.lg) files
  backing up 6–2
  commands to removing entries 16–2

Log File Code Page (-cplog) startup parameter 18–18

Log files
  removing entries 19–24

Log-based replication. *See* Data replication

Log-based site replication 15–2

login.p procedure 10–3

M

-m1 startup parameter 18–27

-m2 startup parameter 18–28

-m3 startup parameter 18–28

-Ma startup parameter 18–29

Machine addressing 5–7

Manual
  syntax notation 2–xxvii

Manual Server (-m2) startup parameter 18–28

Manual Server server type 18–28

Manual, organization of 2–xxiii

MAP option
  INPUT FROM Statement 13–16

MARK BACKEDUP qualifier
  RFUTIL utility 7–14, 19–149

Marking a database backed up 7–14

Maximum Clients per Server (-Ma) startup parameter 18–29

Maximum Dynamic Server Port (-maxport) startup parameter 18–29

Maximum Servers (-Mn) startup parameter 18–32

Maximum Servers per Broker (-Mpb) startup parameter 18–33

-maxport startup parameter 18–29

Media estimates 7–2, 19–11

Memory
  server startup parameters 18–2
  shared 18–34

Memory allocation 14–31, 14–32
  before-image buffers 14–20
  database buffers 14–6, 14–11
  server 14–32

Messages
  displaying descriptions 2–xxxiv

-Mf startup parameter 18–30

-Mi startup parameter 18–31

Minimum Clients per Server (-Mi) startup parameter 18–31

Minimum Dynamic Server (-minport) startup parameter 18–32

-minport startup parameter 18–32

-Mn startup parameter 18–32

Modify Defaults option
  PROMON utility 19–50

Modifying
  bulk loader description file 13–26

Monitor
  database
    main menu 5–15
  PROMON utility 19–25
    R&D options A–1

Monitoring performance 19–116

Monitoring Progress
  buffer activity 14–11
  database fragmentation 14–37
  index use 14–39

Monospaced typeface
  as typographical convention 2–xxvi

Moving
  indexes 9–11
  tables 9–9

-Mp startup parameter 18–33

-Mpb startup parameter 18–33

Multi-threaded servers
  defined 1–9

Multi-user Progress
  starting the server 5–5
  startup commands
    broker 5–5
    server 5–5

Multi-volume databases
adding extents 19–56
creating 19–58
creating structure description files 19–60
operating system limits 3–8
performance advantages 14–6
PROSTRCT utility 19–54
removing extents 19–61
repairing 19–57, 19–63
unlocking 19–66

MVSCH qualifier
PROUTIL utility 19–121

-Mxs startup parameter 18–34

N

-N startup parameter 18–35

-n startup parameter 18–35

Naps 14–5, 14–37

Network Type (-N) startup parameter 18–35

Networks
server startup parameters 18–6

Nickname (-tp) startup parameter
PROUTIL utility 12–8

Nicknames 19–71

No Crash Protection (-i) startup parameter 18–25

No kill users (-bn) startup parameter
PROSHUT command 5–13, 17–12

NO-MAP Option
INPUT FROM Statement 13–16

Non-Progress backups 7–13

NT
backup examples
full backup 7–17
incremental backups 7–20
restoring a full backup 7–30
restoring an incremental backup 7–31

NT Event Log
specifying level of information 18–22

Number of Users (-n) startup parameter 14–32, 18–35

O

Off-line backups 6–6, 7–9

On-line backups 6–5, 7–10, 19–13
after-image extents 6–5

Online startup parameter
PROBKUP utility 7–2, 7–5, 19–10

Operating System
resources 1–12 to 1–13
file descriptors 1–13
processes 1–13
semaphores 1–12
shared memory 1–13
spin locks 1–13

Operating systems
limits 3–8

Overlap factor (-io) startup parameter
PROBKUP utility 19–12

P

Parameter File (-pf) startup parameter 18–36

Partial roll forward 19–150, 19–152

Partial verify (-vp) startup parameter

Passwords 10–2
changing 10–9
encoding with the ENCODE function 10–2
establishing 10–2
validating 10–3

Performance
buffer pool 14–6
direct I/O 14–30
disk I/O 14–6
before image and after-image 14–18
memory 14–31, 14–32
monitoring 14–47, 19–116
multi-volume database advantages 14–6
server startup parameters 18–2
suppressing after-imaging 14–24
using private read-only buffers 14–12

Performance indicators A–63

-pf startup parameter 18–36

-pp startup parameter 18–37

Primary recovery area 1–8
*See also*Before-image area 1–8
size 3–6

Print Code Page (-cpprint) startup parameter 18–18

Private read-only buffers 14–12

PROADSV 19–8

PROAIW command 17–4

PROAIW utility 5–11

PROAPW utility 5–10

PROBIW command 17–6

PROBIW utility 5–11

PROBKUP utility 7–5, 7–10, 19–10
NT examples 7–17
off-line backups 7–9
on-line backups 7–10
UNIX examples 7–4

Procedure library (.pl) files
backing up 6–2

Procedures
examples of 2–xxxi

Processes 14–33
after-image writer (AIW) 17–4
asynchronous page writer 5–10
before-image writer 5–11
before-image writer (BIW) 17–6
cleaning up 17–15

ProControl utility 19–14

PROCOPY 4–16
use to copy system tables, metaschema 4–11

PROCOPY command 4–22

PROCOPY utility 19–17

PRODB utility 4–22, 19–19
creating a database 4–12

PRODEL utility 4–24, 19–22

Progress
broker/server addressing 5–7
database
using a structure file to create 4–10
network addressing 5–7
roll forward recovery 8–2

Progress Explorer 5–2, 17–10
framework 5–2
starting 5–3
utility 19–23

Progress NT Server 19–14

PROLOG
removing log entries 16–2

PROLOG utility 19–24

PROMON utility 19–25
R&D options A–1
accessing A–2
Active Transactions status A–21

AI Log activity A–50
AI Log status A–32
All Processes status A–16
Background Processes status A–25
Backup status A–13
BI Log activity A–48
BI Log status A–31
Blocked Clients status A–20
Buffer Cache activity A–44
Buffer Cache status A–29
Database status A–10
Files status A–26
I/O Operations by File activity A–56
I/O Operations by Type activity A–54
Index activity A–59
Local Batch Clients status A–23
Local Interactive Clients status A–22
Lock Table activity A–52
Lock Table status A–27
Logging Summary status A–30
Main menu A–4
Other activity A–62
Page Writers activity A–46
Record activity A–60
Remote Clients status A–24
Servers activity A–43
Servers status A–14
Shared Memory Segments status A–38
Shared Resources status A–36
Space Allocation activity A–57
Startup Parameters status A–34
Summary activity A–39
Two-Phase Commit status A–33
Resolve Limbo Transaction option A–68
shutting down databases 5–15

Properties file
conmgr.properties file 5–3

PROQUIET command 7–11, 17–7
adjusting the BI Threshold 14–26

PROREST 4–17
LIST qualifier 7–28

PROREST utility 7–23, 7–26, 7–28, 19–52
full verify (-vf) startup parameter 7–26
incremental backups 7–27
-list parameter 7–26, 19–52
NT examples 7–30
partial verify (-vp) startup parameter 7–23, 7–26
UNIX examples 7–29

PROSERVE command 5–7, 17–9

PROSERVE utility 5–5

ProService 19–14

PROSHUT command 17–11

PROSTRCT
ADD qualifier 9–5, 9–7
BUILDDB qualifier 8–25
REMOVE qualifier 9–8

PROSTRCT CREATE 4–2, 4–10, 4–11
example 4–10
output 4–10

PROSTRCT LIST
use to verify database files 4–11

PROSTRCT utility 19–54
LIST qualifier 9–3
log-based replication and 15–7
qualifiers
CREATE 11–5
STATISTICS 9–2
UNLOCK 8–26

PROUTIL
CONV89 Qualifier 19–83
DUMPSPECIFIED qualifier 13–12
ENABLEPDR qualifier 19–100
IDXCOMPACT qualifier 9–11
IDXMOVE qualifier 9–11
MVSCH qualifier 19–121
TABLEMOVE qualifier 9–9
TRUNCATE AREA qualifier 9–8
TRUNCATE BI
qualifier 9–8

PROUTIL CONV89 4–19

PROUTIL utility 19–67
BIGROW qualifier 14–23
CONV89 Qualifier 4–19
DUMP qualifier 13–10

-F startup parameter 8–28
LOAD qualifier 13–21
qualifiers
2PHASE BEGIN 12–7
2PHASE COMMIT 12–16
2PHASE END 12–19
2PHASE MODIFY 12–8
2PHASE RECOVER 12–17
BULKLOAD 13–27
BUSY 7–9, 7–13
IDXBUILD 14–42
IXANALYS 14–39
TABANALYS 14–37
return codes 19–79, 19–101
system crashes 8–17
TRUNCATE BI qualifier 14–21

PROWDOG command 17–15

Purged lock 19–36

Q

Queued lock requests 19–36

R

-r startup parameter 18–37
RFUTIL utility 19–151, 19–153

R&D options
PROMON utility. *See* PROMON utility, R&D options

Raw partitions 2–7

R-code
authorization key 19–90

R-code In Code Page (-cprcodein) startup parameter 18–19

RCODEKEY qualifier
PROUTIL utility 19–122

Rebuilding indexes 14–42, 19–104
phases 14–43, 19–106

RECID field types
dumping and reloading 13–3

Record Locking Table option
PROMON utility 19–35

Record sizes 19–124

Records
modified 14–8

Recovering
from lost or damaged control area 8–25

Recovering bad blocks 7–24

Recovery Log Threshold (-bithold) startup parameter 14–25, 18–14

Recovery plans
developing 6–9
sample 8–7

-red startup parameter
PROBKUP utility 19–12

Redundancy (-red) startup parameter
PROBKUP utility 7–25, 19–12

Relative-path database 1–15, 4–13, 4–23, 9–6

Relative-path,*See*Database

REMOVE qualifier
PROSTRCT utility 19–61

Removing log file entries
PROLOG 16–2

REPAIR qualifier
PROSTRCT utility 19–57, 19–63

Replication. *See* Data replication

Resolve Limbo Transaction option
PROMON utility 19–48, A–68

Resolving limbo transactions
case study 12–19
PROMON 12–11
PROUTIL 12–15

Resource locking
UNIX 14–33
kernel parameters 14–33

Restoring
backup 7–28

Restoring databases 7–25, 8–17, 8–18
recovering bad blocks 7–24

Return codes
Progress/UTILITIES utility 19–79
Progress/UTILITY command 19–101
PROUTIL utility 19–79, 19–101

RFUTIL utility 19–137
AIMAGE EXTENT FULL 19–143
AIMAGE TRUNCATE 14–29
log-based replication and 15–8
qualifiers
MARK BACKEDUP 7–14
ROLL FORWARD 8–19, 8–20
system crashes 8–17

RM chains
analyzing 19–81

ROLL FORWARD qualifier
RFUTIL utility 8–19, 8–20, 19–150

Roll forward recovery 8–2, 8–4
after-imaging 8–4
before-imaging 8–3
preparations for using 8–6
RFUTIL utility 19–137
sample scenario 8–11

Roll-forward recovery
do’s and don’ts 6–13

Running
user reports 10–10

S

-S startup parameter 5–7, 18–38
PROSHUT command 5–13, 17–12

-s startup parameter
PROCOPY utility 19–17

Scan
after-image file 19–146

Scan (-scan) startup parameter
PROBKUP utility 6–6, 19–12

Schema
adding to a void database 4–11

Schema area 1–8

Schema security
establishing 10–11
freezing and unfreezing tables 10–12

Secondary Login Broker (-m3) startup parameter 18–28

Secondary Login Broker server type 18–28

Security
connection 10–5
establishing authentication 10–3
schema 10–11

Security administration
adding a user to user list 10–7
changing password 10–9
deleting a user from user list 10–9
designating a security administrator 10–5
freezing and unfreezing tables 10–12
running a user report 10–10
validating user ID and password 10–3

Semaphore Sets (-semsets)
broker startup parameter 14–36

Semaphore Sets (-semsets) startup parameter 18–39

Semaphores 14–34
allocating 14–36
defined 1–12
number used 19–42
-semsets broker startup parameter 14–36

-semsets startup parameter 18–39

Sequences
dumping 13–8
dumping and loading 13–24

_seqvals.d file 13–8, 13–24

Server
multi-threaded 1–9

Server Group (-servergroup) startup parameter 18–38

Server startup parameters
buffers 18–2
networks 18–6
performance 18–2
server type specification 18–4
statistic collection specification 18–5

Server types 18–27

servergroup 17–9

-servergroup startup parameter 17–9, 18–38

Servers
addressing 5–7
controlling number spawned for one protocol 18–33
multi-threaded 1–9
shutting down 5–12
starting 5–7

Servers per Protocol (-Mp) startup parameter 18–33

Server-type startup parameters 18–4

Service Name (-S) startup parameter 5–7, 18–38
PROSHUT 5–13, 17–12

SETUSERID function 10–3

Shared memory
and on-line backups 19–13
defined 1–13
file descriptors 1–13
processes 1–13
semaphores 1–12
shared memory segments 1–13
spin locks 1–13
status 19–92
status information 19–92
version number 19–42

Shared Resources option
PROMON utility 19–41

Shared-memory Overflow Size (-Mxs) startup parameter 18–34

Shut Down Database option
PROMON utility 19–45

Shutdown commands 17–11

Shutting down
AIWs 5–11
APWs 5–10
BIWs 5–11
brokers 5–12
databases 5–12
PROMON 5–15
PROSHUT 5–12
servers 5–12

Silent (-s) startup parameter
PROCOPY utility 19–17

Single-user mode
and on-line backups 19–13

Single-user Progress
on-line backups 6–5

Site replication 15–2

Sizes
of databases 19–44

Sorting
disk space required 2–6

Spawning servers
manually 18–28

Spin Lock Retries (-spin) startup parameter 18–40

Spin locks 14–5, 14–37
algorithm 14–5, 14–37

-spin startup parameter 18–40

Spinning 14–5, 14–37

SQL views
dumping file contents 13–18
loading file contents 13–24

SQLDUMP Utility 19–154

SQLLOAD Utility 19–159

SQLSchema Utility 19–164

Starting Progress
AIWs 5–11
APWs 5–10
BIWs 5–11
multi-user
with a server 5–5

Startup commands
summary 17–3
syntax 17–2

Startup parameters
After-image Buffers (-aibufs) 14–28
after-image stall (-aistall) 11–7
Before-image Buffers (-bibufs) 14–20
blocking factor (-bf)
PROBKUP utility 7–25
Buffer Size (-B) 14–11
buffers 18–2
changing defaults 18–2
database server international 18–4
Delayed BI File Write (-Mf) 14–25
displaying settings 19–41
Force Access (-F)
PROUTIL utility 8–28
Hash Table Entries (-hash) 14–12
memory 18–2
networks 18–6
Number of Users (-n) 14–32
performance 18–2
redundancy (-red)
PROBKUP utility 7–25
server type specification 18–4
statistic collection specification 18–5

Statistic Collection startup parameters 18–5

Statistics
for storage utilization 9–2
PROMON utility
buffer pool access 19–32
Locking and Waiting option 19–27, 19–30
PROSTRCT utility 19–64

STATISTICS qualifier
PROSTRCT utility 19–64

Storage
logical 1–6
physical 1–5
temporary 2–6
utilization statistics 9–2
variable-length technique 2–3

Storage areas 3–2
adding 9–5
adding extents 9–7
after-image area 1–8
creating 4–8
defining 4–7
location 2–3
number of 3–2
primary recovery area
size 3–6
removing 9–8
size of 3–2
transaction log area 1–8, 12–9
truncating 19–130

Storage areas,*See Areas*

Storage objects 1–7

Storing databases
raw partitions 2–7

Stream Code Page (-cpstream) startup parameter 18–20

Structure description file
creating 4–3
defining storage areas 4–7
example 4–3
specifying pathnames 4–7
updating information 9–3

Structure description files 19–58
appending 19–56
bnf syntax 4–5
creating 19–60
extent length 4–3, 4–9
extent pathname 4–3
extent type 4–3, 4–9
f token 4–9
PROSTRCT utility 8–25, 19–55, 19–56, 19–57, 19–63
repairing 19–57, 19–63

Structure files
backing up 6–2
unlocking damaged 8–26

Synchronous replication 15–2

Syntax notation 2–xxvii

System crash
backup procedures 8–18
loss of the after-image file 8–19
loss of the database backup 8–20
loss of the transaction log file 8–21
using PROUTIL and RFUTIL 8–17
using utilities 8–17
while running Progress 8–18

T

TABANALYS qualifier
PROUTIL utility 14–37, 19–124

Table Base (-basetable) startup parameter 18–12

Table Limit (-tablelimit) startup parameter 18–41

TABLEMOVE qualifier 9–9
PROUTIL utility 19–127

-tablerangesize startup parameter 18–41

Tables
adding 9–9
dumping 8–27
dumping and reloading. *See* Databases, dumping and loading
freezing and unfreezing definitions 10–12
moving 9–9
user tables
dumping contents 13–17
loading contents 13–23
using one in multiple databases 13–32

TCP/IP networks
shutdown command parameters 5–15
shutting down the server 5–15
specifying host name of remote machine 5–15

Temporary storage 2–6

Terminal behavior
PROMON utility 19–50

Terminal Code Page (-cpterm) startup parameter 18–21

Text files
converting character sets 19–87

Threshold Stall (-bistall) startup parameter 14–26, 18–13

Time stamp
for file data definition 10–11

-tp startup parameter
PROUTIL 2PHASE BEGIN utility 19–71
PROUTIL 2PHASE MODIFY utility 19–75

Transaction locks 19–31

Transaction log (.tl) files 12–6
backing up 6–2

Transaction log area 12–9

Transaction log file disks 8–23

Transactions
  distributed 12–2
  number 12–3
  status 12–3
  two-phase commit 8–6, 12–2

Transactions Control option
  PROMON utility 19–46

Transfer speed
  PROBKUP 7–3, 19–11

Trigger-based replication 15–2

Triggers
  dumping and loading 13–2

TRUNCATE AREA qualifier
  PROUTIL utility 19–130

TRUNCATE BI qualifier
  PROUTIL utility 14–21, 19–132

Truncating
  after-image file 19–145, 19–147
  application data storage areas 19–130
  before-image files 8–24, 19–132
  log files 19–24

Tuning
  BIWs 14–20
  database buffers 14–11
  hash table 14–12
  memory
    server 14–32
  memory allocation 14–31, 14–32
  resource locking
    UNIX 14–33

2PHASE BEGIN qualifier
  PROUTIL utility 19–71

Two-phase commit 12–1
  after-imaging 12–6
  algorithm 12–4
  case study 12–19
  changing nickname 19–75
  changing priority 19–75
  coordinator database 19–71
  data replication and 15–2
  deactivating 12–19
  disabling 19–74
  enabling 12–7, 19–71
  implementation 12–3
  introduction 8–6, 12–2
  limbo transactions 19–73
  nicknames 12–7, 19–71
  phases 12–3
  transaction log area 12–9

2PHASE COMMIT qualifier
  PROUTIL utility 19–73

2PHASE END qualifier
  PROUTIL utility 19–74

2PHASE MODIFY qualifier
  PROUTIL utility 19–75

2PHASE RECOVER qualifier
  PROUTIL utility 19–76

Typographical conventions 2–xxvi

U

Unconditional shutdown menu option
  PROMON utility 19–45

UNIX 1–12
  backup examples
    full backups 7–4
    incremental backups 7–14
    restoring a full backup 7–29
    restoring an incremental backup 7–29
  backup utilities 6–4, 11–10
  file descriptors 1–13
  kernel parameters
    resource locking 14–33
  processes 1–13
  semaphores 1–12, 14–34
  shared memory 1–13
  spin locks 1–13

UNLOCK qualifier
  PROSTRCT utility 19–66

UPDATEVST qualifier
PROUTIL utility 20–2

Upgrade requests 19–36

Usage reporting
system resources 19–41

User IDs 10–2
establishing 10–2
validating 10–3

User lists 10–2

User report
running 10–10

Users
maximum number 18–35

Utilities
Autoconvert 4–24
backup 6–4, 11–10
PROAIW 5–11
PROAPW 5–10
PROBIW 5–11
PROBKUP 7–5, 7–10
PRODB 4–12
PRODEL 4–24
PROMON 5–15
PROREST 7–23, 7–26, 7–28
PROSERVE 5–5, 5–8
PROSHUT 5–12
PROSTRCT
CREATE qualifier 11–5
LIST qualifier 9–3
STATISTICS qualifier 9–2
UNLOCK qualifier 8–26
PROUTIL
BULKLOAD qualifier 13–27
RFUTIL
MARK BACKEDUP qualifier 7–14
ROLL FORWARD qualifier 8–19, 8–20
RFUTIL AIMAGE EXTENT FULL 19–143
system crashes 8–17

Utilization
index 14–39
analyzing 14–39

V

Validating user ID and password 10–3

Variable block sizes 19–58

Variable-length extents 4–9
after-image 11–3
on-line backups 6–5

Variable-length *See* Extents

Verbose (-verbose) startup parameter
PROBKUP utility 7–3, 19–11
RFUTIL utility 19–146

-verbose startup parameter
RFUTIL utility 19–146

Verifying a backup
PROREST utility 7–23

-vf startup parameter
PROREST utility 19–52

Virtual system tables (VST) 9–12, 14–47, 20–1
descriptions 20–8
obtaining status of utilities 9–12
summaries 20–2
updating 19–134, 20–2
updating access to 19–134, 20–2
using 20–1

VLM Page Table Entry Optimization (-Mpte) startup parameter 18–34

VMS
backup utilities 6–4, 11–10

Volume Size (-vs) startup parameter
PROBKUP utility 7–3, 19–11

-vp startup parameter
PROREST utility 19–52

-vs startup parameter
PROBKUP utility 7–3, 19–11

W

Watchdog utility
See also PROWDOG utility

WBREAK-COMPILER qualifier
PROUTIL utility 19–135

Word break table 19–135