

Progress/400
Product Guide

Copyright© 2000 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®, Progress Results®, and WebSpeed® are registered trademarks of Progress Software Corporation.

Apptivity™, AppServer™, ProVision™, ProVision™ Plus, SmartObjects™, SonicMQ™, and all other Progress product names are trademarks of Progress Software Corporation.

Progress Software Corporation acknowledges the use of Raster Imaging Technology copyrighted by Snowbound Software 1993-1997. Progress® SonicMQ™ contains the IBM® XML Parser for Java Edition and the IBM® Runtime Environment for Windows®, Java™ Technology Edition Version 1.1.8 Runtime Modules.

Copyright© 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 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.

All other company and product names are the trademarks or registered trademarks of their respective companies.

June 2000

Product Code: 4608

Item Number: 68091;9.1B

Contents

Preface . . . . . **xvii**
Purpose xvii
Audience xvii
Organization of This Manual xviii
Typographical Conventions xx
Syntax Notation xxi
Progress Messages xxiv
Other Useful Documentation xxvi
Getting Started xxvi
Development Tools xxvii
Reporting Tools xxviii
4GL xxix
Database xxx
DataServers xxx
SQL-89/Open Access xxx
SQL-92 xxxi
Deployment xxxi
WebSpeed xxxii
Reference xxxii

1. Introduction **1–1**
1.1 Progress/400 Architecture 1–2
1.1.1 Client/Server, Web-based, or N-tier Architecture 1–2
1.1.2 Native 4GL and Progress/400 AppServer Architecture 1–3
1.1.3 Inside Progress/400 1–4
1.2 Progress/400 Objects 1–5
1.2.1 A Dictionary Library 1–5
1.2.2 The Server Schema 1–5
1.2.3 Progress/400 Dictionary Objects 1–7

1.2.4 The Schema Holder 1–7
1.2.5 The Schema Image 1–7
1.2.6 Demonstration Databases 1–8
1.3 Progress/400 Utilities 1–8
1.4 Using Progress/400 1–9
1.4.1 Accessing Existing DB2/400 Database Files 1–9
1.4.2 Migrating Progress Databases to the AS/400 1–9
1.4.3 Upgrading to the Current Progress/400 Product 1–10
1.4.4 Running Progress Applications on the AS/400 1–10
1.4.5 Security 1–11
1.5 Where to Find Further Information 1–11
1.5.1 This Manual 1–11
1.5.2 Related Topics 1–13

2. Common Product Information 2–1
2.1 Database Design Issues 2–2
2.1.1 DB2/400 and Progress Objects and Terminology 2–2
2.1.2 Naming Conventions 2–3
2.1.3 Indexes and Keys 2–4
2.1.4 Triggers 2–5
2.1.5 Views 2–5
2.1.6 Virtual Tables 2–5
2.1.7 Multiple Members in Physical Files 2–6
2.1.8 Progress/400 and Distributed Data Management 2–7
2.2 Data Types 2–10
2.2.1 DB2/400-to-Progress Data Type Mapping 2–10
2.2.2 Progress-to-DB2/400 Data Type Mapping 2–11
2.2.3 Progress Unknown Value Support 2–14
2.3 Transactions 2–16
2.3.1 Record Creation 2–17
2.3.2 Record Locking 2–18
2.4 Queries 2–22
2.4.1 Joined Logical Files and Progress Queries 2–22
2.4.2 Record Prefetch 2–23
2.5 Field Lists 2–23
2.6 Language Restrictions 2–25
2.6.1 ALTER TABLE, CREATE INDEX, CREATE TABLE, CREATE VIEW, DROP INDEX, DROP TABLE, DROP VIEW Statements 2–25
2.6.2 DBNAME Function 2–25
2.6.3 GRANT Statement 2–25
2.6.4 ROWID Function 2–26
2.6.5 RECID Function 2–26
2.6.6 REVOKE Statement 2–27

2.6.7 SETUSERID Function 2–27
2.6.8 USERID Function. 2–27
2.7 Application Security 2–27
2.7.1 Setting Up a User Permission File on the AS/400 2–28
2.7.2 Defining a User ID and Password at Startup 2–28
2.8 Progress/400 Word Index Support 2–28
2.8.1 Setting Up and Using Word Indexes 2–29
2.8.2 How Progress/400 Maintains Word Indexes 2–31
2.8.3 Coding Issues 2–34

3. Creating the Progress/400 Environment 3–1
3.1 Overview 3–2
3.2 Accessing an Existing DB2/400 Database 3–3
3.2.1 Creating the Environment (DUPPRODB). 3–3
3.2.2 Changing Data Definitions (CHGPRODCT). 3–4
3.2.3 The Client Schema Holder. 3–5
3.2.4 Maintaining DB2/400 Data Definitions 3–9
3.2.5 Programming Considerations. 3–10
3.3 Moving a Progress Database to the AS/400 3–10
3.3.1 Creating the Environment (DUPPRODB). 3–11
3.3.2 Dumping Data Definitions and Data. 3–12
3.3.3 The Client Schema Holder 3–13
3.3.4 Maintaining Data Definitions on the AS/400 3–18
3.3.5 Programming Considerations 3–19

4. Remote Access to Progress/400 DataServer 4–1
4.1 How Progress/400 Accesses DB2/400 Database Files 4–2
4.2 Remote Client Connectivity 4–3
4.2.1 TCP/IP Communications Protocol 4–3
4.2.2 SNA Communications Protocol 4–5
4.2.3 Connection Parameters 4–6
4.2.4 DATALIB Argument Usage Notes 4–10
4.2.5 Connection Techniques 4–15
4.2.6 General Connection Considerations 4–17
4.2.7 Connection Troubleshooting 4–19

5. Preparing to Use AS/400-based Clients 5–1
5.1 Overview 5–2
5.2 Understanding the Integrated File System 5–2
5.2.1 The ROOT File System Under the IFS 5–2
5.2.2 Absolute and Relative Paths 5–3
5.2.3 Accessing QSYS.LIB Files. 5–3
5.3 Task List 5–4

5.4 Creating a Schema Image 5–4
5.4.1 New DB2/400 Database 5–5
5.4.2 Existing Server Schema 5–5
5.5 Executing Progress Code from the Native Clients 5–6
5.5.1 Preparing Progress 4GL Procedures for the AS/400 5–6
5.5.2 Transferring Progress 4GL Procedures to the AS/400 5–10
5.5.3 PROPATH Construction Rules 5–11
5.5.4 Compiling P-code on the AS/400 5–11
5.6 Internationalizing the Native Clients 5–12
5.7 Internationalization Startup Parameters 5–12

6. Using the Progress/400 Native 4GL Client 6–1
6.1 Overview 6–2
6.2 Running Progress Applications Using the Native 4GL Client 6–2
6.3 Using QCMD to Start the Native 4GL Client Remotely 6–2
6.4 Passing Parameters to the Native 4GL Client 6–3
6.5 Executing Triggers 6–3
6.6 Preserving the Working Directory 6–3
6.7 Calling Progress 4GL Programs from OS/400 HLL Programs 6–4
6.7.1 The PROCALL Program 6–4

7. Using the Progress/400 AppServer 7–1
7.1 Overview 7–2
7.1.1 The Progress/400 AdminServer 7–2
7.1.2 The Progress/400 AppServer Instance 7–3
7.1.3 The Progress/400 NameServer 7–4
7.2 Task List 7–5
7.3 Managing the Progress/400 AppServer Product 7–7
7.3.1 Starting the Progress/400 AdminServer 7–8
7.3.2 Modifying the Configurations. 7–8
7.3.3 Starting the Progress/400 NameServer 7–9
7.3.4 Starting the Progress/400 AppServer 7–9
7.3.5 Managing the Progress/400 NameServer and Progress/400 AppServer. 7–9
7.3.6 Stopping the Progress/400 AdminServer 7–9
7.3.7 Progress/400 AppServer Instance Job Information 7–10
7.3.8 Progress/400 AppServer Instance Environment 7–11
7.3.9 Naming Multiple Servers 7–11
7.3.10 Naming Conventions 7–12

8. System Administration 8–1
8.1 Working with Progress Jobs 8–2
8.2 Security 8–2
8.2.1 Database Security 8–2
8.2.2 AppServer Security 8–4
8.3 Transaction Control 8–4
8.3.1 Local Before-imaging 8–4
8.3.2 Specifying Transaction Control Techniques. 8–5
8.3.3 Recovery 8–6
8.3.4 Journaling 8–6
8.4 Code Pages and Collation Support 8–8
8.4.1 Code Page Overview 8–9
8.4.2 Setting Up Collation for Progress/400 Databases 8–13
8.4.3 User-defined Collation Tables 8–17
8.5 Controlling DataServer Performance 8–21
8.5.1 Query Processing 8–21
8.5.2 Managing Network Traffic 8–22
8.5.3 Loading Definition Files 8–22
8.6 Progress Settings File (PROSET) 8–22
8.7 Retaining Progress Attributes 8–28
8.8 Progress/400 Reserved Words 8–30
8.9 Creating Test and Production Environments 8–31

9. Remote Client DB2/400 Utilities 9–1
9.1 DataServer Utilities in the Data Administration Tool 9–2
9.1.1 Create DB2/400 DataServer Schema 9–2
9.1.2 Synchronizing a Client Schema Holder 9–3
9.1.3 Changing Connection Information 9–4
9.1.4 Deleting a Client Schema Holder 9–5
9.2 Progress/400 Data Dictionary 9–5
9.2.1 Adding a Table 9–7
9.2.2 Modifying a Table 9–8
9.2.3 Deleting a Table 9–9
9.2.4 Adding a Field 9–10
9.2.5 Defining Field Length 9–12
9.2.6 Field Format Length 9–12
9.2.7 Modifying a Field 9–13
9.2.8 Deleting a Field 9–14
9.2.9 Adding an Index 9–14
9.2.10 Modifying an Index 9–16
9.2.11 Deleting an Index 9–17
9.2.12 Adding a Sequence 9–17
9.2.13 Modifying a Sequence 9–18
9.2.14 Deleting a Sequence 9–18

9.2.15 Adding a Procedure 9–19
9.2.16 Modifying a Procedure 9–20
9.2.17 Deleting a Procedure 9–21
9.2.18 Adding a Parameter 9–21
9.2.19 Modifying a Parameter 9–23
9.2.20 Deleting a Parameter 9–24
9.3 Progress/400 Data Dictionary Administration Utilities 9–24
9.3.1 Dumping DB2/400 Data Definitions 9–25
9.3.2 Dumping Data 9–26
9.3.3 Dumping Sequence Definitions 9–27
9.3.4 Dumping Sequence Current Values 9–28
9.3.5 Creating Incremental .df Files 9–28
9.3.6 Loading Data Definitions 9–30
9.3.7 Loading Data 9–32
9.3.8 Loading Sequence Current Values 9–33
9.3.9 Reconstructing Bad Load Records 9–33
9.3.10 Synchronizing the Client 9–33
9.3.11 Freezing/Unfreezing a Table 9–35

10. AS/400 Utilities 10–1
10.1 Progress/400 Utilities Grouped by Subject Area 10–2
10.2 Running Server Utilities 10–3
10.2.1 Running Server Utilities from the Menu 10–4
10.2.2 Running Server Utilities from the Command Line 10–4
10.2.3 Accessing Online Help 10–5
10.3 Progress/400 AppServer Commands 10–5
10.3.1 End Progress/400 AdminServer (ENDADMSVR) 10–6
10.3.2 Set Progress/400 Environment (SETPROENV) 10–6
10.3.3 Start Progress/400 AdminServer (STRADMSVR) 10–7
10.4 Progress/400 Dictionary Library & Utility Commands 10–8
10.4.1 Change Progress/400 Dictionary Utility (CHGPRODCT) 10–8
10.4.2 Change Progress/400 Defaults Utility (CHGPRODFT) 10–11
10.4.3 Create Progress/400 Lock Table (CRTPROLKT) 10–13
10.4.4 Create Schema Image (CRTSCHIMG) 10–13
10.4.5 Convert Progress/400 Dictionary (CVTPRODCT) 10–14
10.4.6 Convert Server Schema (CVTSRVSCH) 10–15
10.4.7 Duplicate Progress/400 Database (DUPPRODB) 10–15
10.4.8 DUMP DATA (DMPPRODTA) 10–21
10.4.9 LOAD DATA (LODPRODTA) 10–22
10.4.10 Remove Server Schema Entry (RMVSCHE) 10–23
10.4.11 Synchronize Schema Image (SYNSCHIMG) 10–25

10.5 Progress/400 Word Index Support Commands 10–26
10.5.1 End Word Index Support Processor (ENDWISPRC) 10–26
10.5.2 Generate Word Index (GENWRDIDX) 10–26
10.5.3 Replace Dictionary Word Break Table (RPLDCTWBT) 10–27
10.5.4 Start Word Index Support Processor (STRWISPRC) 10–27
10.5.5 Update Progress Word Break Table (UPDPROWBT) 10–28
10.5.6 Update Word Index Support Triggers (UPDWISTRG) 10–30
10.6 Progress/400 TCP/IP Network Commands 10–31
10.6.1 End Progress Network (ENDPRONET) 10–31
10.6.2 Manage Progress/400 Networking (MNGPRONET) 10–32
10.6.3 Start Progress/400 Networking (STRPRONET) 10–32
10.6.4 Work with Progress Jobs (WRKPROJOB) 10–33
10.7 Progress/400 Native 4GL Client Commands 10–34
10.7.1 Start Native 4GL Client (STRPROCLI) 10–35

11. Progress 4GL Interfaces to OS/400 Languages and Objects 11–1
11.1 External Programming Interface (EPI) 11–2
11.1.1 General Syntax 11–2
11.1.2 EPI CALL Command 11–3
11.2 Executing OS/400 Commands 11–8
11.2.1 QCMD Functions 11–8
11.2.2 Running RPG Programs with QCMD 11–10
11.2.3 Executing the SBMJOB Command 11–10
11.2.4 Accessing Data from Multiple Members 11–12
11.2.5 SBMJOB Parameters 11–13
11.3 Progress/400 Stored Procedure Support 11–15
11.3.1 Running a Stored Procedure 11–18
11.4 OS/400 Object Access 11–20
11.4.1 Data Area Support 11–21
11.4.2 User Space Support 11–27
11.4.3 Data Queue Support 11–32
11.4.4 Using the LOOKUP and SUBSTRING Functions with Send Data 11–35

A. Tutorials for Managing Your Dictionary Library A–1
A.1 Creating a New DB2/400 Database with Progress/400 . . . . . A–2
A.1.1 Creating a New Database with No Existing Data A–2
A.1.2 Creating a Database Using Existing DB2/400 Data A–3
A.2 Creating a Copy of a Schema and Data A–5
A.2.1 Copying a Single Library or ALTLIB Database A–5
A.2.2 Creating a Copy of an ALTLIB Database Structure A–6
A.3 Modifying DB2/400 Data Definitions with Progress/400 . . . . . A–9
A.3.1 Modifying Data Definitions Using the Progress/400 Data Dictionary A–9

A.3.2 Modifying Data Definitions Using an AS/400 Tool A–10

B. Legacy Support and the Progress Unknown Value B–1
B.1 Legacy Unknown Value B–2
B.2 DUPPRODB Legacy Unknown Value Support B–3
B.3 Converting from Legacy Unknown Values to SQL NULL Support B–3
B.4 Using the Windows Client to Manage SQL NULL Value Assignments . . B–4
B.5 How Progress/400 Handles SQL NULL and Legacy Unknown Value Support B–5

C. Useful OS/400 Commands C–1
C.1 OS/400 CL Command List C–2

Glossary Glossary–1

Index Index–1

Figures

Figure 1–1: Progress/400 Client/Server Architecture 1–2
Figure 1–2: Progress/400 Host-based Architecture 1–3
Figure 1–3: Progress/400 Internals 1–4
Figure 2–1: DDM Architecture 2–8
Figure 3–1: The Progress/400 Environment 3–2
Figure 4–1: How Progress/400 Accesses DB2/400 Database Files 4–2
Figure 4–2: TCP/IP Communications Jobs 4–3
Figure 4–3: SNA Communications Jobs 4–6
Figure 7–1: Progress/400 AdminServer Architecture 7–3
Figure 7–2: Progress/400 AppServer Instance Architecture 7–4
Figure 7–3: NameServer Architecture 7–5
Figure 8–1: Server and Client Code Page Relationships 8–10
Figure 8–2: DUPPRODB Prompts and Responses 8–15
Figure 8–3: Collation Table Example 8–19
Figure 8–4: Specifying a Sort Order in a Collation Table 8–20
Figure 8–5: How DUPPRODB Works with the *FULLCPY Option 8–33
Figure 8–6: How DUPPRODB Works with the *DCTONLY Option 8–34
Figure 9–1: Progress/400 Data Dictionary Window 9–6

Tables

Table 1–1: Progress/400 Server Schema Files 1–6
Table 1–2: How to Use This Manual 1–12
Table 1–3: Progress-related Topics 1–13
Table 2–1: Progress and DB2/400 Objects 2–2
Table 2–2: DB2/400-to-Progress Data Type Mapping 2–10
Table 2–3: DB2/400 Date Formats 2–13
Table 2–4: Progress/400 Data Dictionary Options for SQL NULL and Unknown Value Assignments 2–15
Table 2–5: Standard Progress and Progress/400 Transactions 2–16
Table 2–6: CRTPROLKT Parameters 2–19
Table 2–7: OS/400 Impact on Progress Applications 2–21
Table 2–8: Word Index Corruption Recovery Procedures 2–33
Table 3–1: AS/400 Network Processes 3–5
Table 3–2: Network Connection Parameters 3–8
Table 3–3: AS/400 Network Processes 3–12
Table 3–4: Network Connection Parameters 3–16
Table 4–1: Progress/400 Connection Parameters 4–7
Table 4–2: DataServer (-Dsrv) Arguments 4–9
Table 4–3: Network Type (-N) Arguments 4–14
Table 4–4: Server Name (-Sn) Options 4–15
Table 4–5: Connection Failure Responses 4–19
Table 4–6: Connection Failure Error Messages 4–20
Table 5–1: QSYS.LIB and IFS Path Resolutions 5–3
Table 5–2: DUPPRODB Parameter 5–5
Table 5–3: Integrated File System INPUT FROM Syntax 5–7
Table 5–4: Integrated File System OUTPUT TO Syntax 5–8
Table 5–5: QCMD Commands for File Transfer 5–10
Table 5–6: Defaults for -cp* Startup Parameters 5–12
Table 8–1: DataServer (-Dsrv) Arguments 8–5
Table 8–2: CL Commands for Journaling 8–7
Table 8–3: Code Pages and Corresponding ALTSEQ Tables 8–16
Table 8–4: PROSET Settings 8–23
Table 8–5: CHGPRODCT PROATR Behavior 8–29
Table 9–1: DataServer Utilities 9–2
Table 9–2: DB2/400 File Attributes 9–8
Table 9–3: DB2/400 Data Types with Progress Equivalents 9–22
Table 9–4: Progress-to-DB2/400 Data Type Mapping 9–31
Table 10–1: ENDADMSVR Parameters 10–6
Table 10–2: SETPROENV Parameter 10–6
Table 10–3: Environment Variables Affected by SETPROENV 10–7
Table 10–4: CHGPRODCT Parameters 10–9
Table 10–5: CRTPROLKT Parameters 10–13
Table 10–6: CRTSCHIMG Parameter 10–13

Table 10–7: CVTPRODCT Parameters 10–14
Table 10–8: CVTSRVSCH Parameter 10–15
Table 10–9: DUPPRODB Parameters 10–16
Table 10–10: Progress/400 Server Schema Files 10–20
Table 10–11: Progress/400 Server Schema Objects 10–21
Table 10–12: DMPPRODTA Parameters 10–22
Table 10–13: LODPRODTA Parameters 10–22
Table 10–14: RMVSCHE Parameters 10–23
Table 10–15: SYNSCHIMG Parameter 10–25
Table 10–16: GENWRDIDX Parameters 10–27
Table 10–17: RPLDCTWBT Parameters 10–27
Table 10–18: STRWISPRC Parameters 10–28
Table 10–19: UPDPROWBT Parameters 10–29
Table 10–20: UPDWISTRG Parameters 10–30
Table 10–21: ENDPRONET Parameters 10–31
Table 10–22: MNGPRONET Parameters 10–32
Table 10–23: STRPRONET *TCP Parameters 10–32
Table 10–24: WRKPROJOB Parameter 10–34
Table 10–25: STRPROCLI Parameters 10–35
Table 10–26: Startup Parameters for STRPROCLI 10–36
Table 11–1: OS-ERROR Values for EPI Command 11–2
Table 11–2: Mapping Progress to OS/400 Data Types 11–5
Table 11–3: USE Parameter 11–6
Table 11–4: SBMJOB Parameters 11–13
Table 11–5: Information Contained in a Stored Procedure 11–16
Table 11–6: Stored Procedure Argument Data Types 11–17
Table 11–7: Functions for Stored Procedure Return Codes 11–18
Table 11–8: Stored Procedure Implementation Restrictions 11–20
Table 11–9: CHANGE DATA AREA (chgdara.p) Parameters 11–22
Table 11–10: CHANGE DATA AREA (chgdara.p) Return Values 11–23
Table 11–11: RETRIEVE DATA AREA (rtvdara.p) Parameters 11–24
Table 11–12: RETRIEVE DATA AREA (rtvdara.p) Return Values 11–25
Table 11–13: CHANGE USER SPACE (chguspc.p) Parameters 11–28
Table 11–14: CHANGE USER SPACE (chguspc.p) Return Values 11–29
Table 11–15: RETRIEVE USER SPACE (rtvuspc.p) Parameters 11–30
Table 11–16: RETRIEVE USER SPACE (rtvuspc.p) Return Values 11–31
Table 11–17: SEND DATA QUEUE (snddtaqe.p) Parameters 11–33
Table 11–18: SEND DATA QUEUE (snddtaqe.p) Return Values 11–35
Table 11–19: RECEIVE DATA QUEUE (rcvdtaqe.p) Parameters 11–36
Table 11–20: RECEIVE DATA QUEUE (rcvdtaqe.p) Return Values 11–38
Table A–1: DUPPRODB Parameter Values Required for No Existing Data A–2

Table A–2: DUPPRODB Parameter Values Required for Using Existing Data A–3
Table A–3: CHGPRODCT Parameter Values for Using Existing Data A–4
Table A–4: DUPPRODB Parameter Values for a Library Copy A–6
Table A–5: DUPPRODB Parameter Values for Empty Server Schema A–7
Table A–6: CHGPRODCT Parameter Values for ALTLIB Database Structure A–8
Table A–7: CHGPRODCT Selections to Update Server Schema A–11
Table B–1: The Legacy Unknown Value Implementation by Data Type B–2
Table B–2: Legacy Unknown Value Defaults for DB2/400 Date Formats B–3
Table B–3: Progress/400 Data Dictionary Options for SQL NULL and Unknown Value Assignments B–5
Table B–4: SQL and 4GL Query Results with NULL Capable and ALWPROUNK Settings B–5
Table C–1: Useful OS/400 Commands C–2

Procedures

PGM1.P 4–11
PGMA1.P 4–11
CL Program That Calls a Progress Program 6–4
RPG IV Program That Calls a Progress Program 6–5
Progress Program sample.p 6–5
Progress 4GL Program TestCall.P 11–7
CL Program TESTCLP 11–7

Preface

Purpose

This manual explains how to use the Progress/400 products that include the Progress/400 DataServer, the Native 4GL Client, and the Progress/400 AppServer. This manual discusses database design and programming issues to consider when creating applications that access the Progress and DB2/400 database management systems. Additionally, it describes the client and server utilities that support the Progress/400 products. Note that this manual is intended specifically for Version 9.1 and later of the Progress/400 Product.

Throughout this document, DB2/400 refers to any database on the AS/400 that was created or is maintained using DDS, IDDU, or SQL/400. Progress Software Corporation uses this name to be consistent with IBM documentation.

If you are a Progress/400 customer, use this manual and the standard Progress (MS-Windows, OS/2, and UNIX) documentation set for your specific Progress needs.

Audience

The *Progress/400 Product Guide* addresses these audiences:

- Progress application developers familiar with client/server architecture
- AS/400 developers
- DB2/400 database administrators (DBAs)

Organization of This Manual

Chapter 1, "Introduction"

Describes the basic architecture of the Progress/400 products and presents guidelines for using the Progress/400 DataServer, the Native 4GL Client, and the Progress/400 AppServer.

Chapter 2, "Common Product Information"

Explains how the Progress 4GL works with the Progress/400 products. It includes information on data-type mapping and differences in the Progress 4GL when used against a DB2/400 database.

Chapter 3, "Creating the Progress/400 Environment"

Provides step-by-step instructions for setting up the server schema that allows you to access DB2/400 database files with Progress applications through the Progress/400 DataServer. This chapter also provides step-by-step instructions for moving a Progress database to the AS/400.

Chapter 4, "Remote Access to Progress/400 DataServer"

Describes the supported network protocols-SNA and TCP/IP-and provides instructions for starting and maintaining the OS/400 communications jobs that these protocols require. This chapter also describes how to connect to the schema holder and the DB2/400 database in client/server mode. It includes information on Progress connection parameters and connection troubleshooting.

Chapter 5, "Preparing to Use AS/400-based Clients"

Provides information on how to set up and use the AS/400-based clients. Topics discussed here include the Integrated File System (IFS), creating a schema image, executing Progress code on the AS/400, and internationalizing the native clients.

Chapter 6, "Using the Progress/400 Native 4GL Client"

Explains starting, passing parameters, and executing triggers on the Native 4GL Client.

Chapter 7, "Using the Progress/400 AppServer"

Explains Progress/400 AppServer product-specific configuration, management, programming, and installation considerations.

Chapter 8, “System Administration”

Describes how Progress uses the native AS/400 facilities to handle database security, recovery, transaction undo, and locking. It also explains how to provide foreign character support, tune the DataServer environment, submit batch jobs from the Progress client, and duplicate a Progress/400 Dictionary Library to create a production or test environment.

Chapter 9, “Remote Client DB2/400 Utilities”

Describes the utilities that you use to create and maintain the client schema holder on the client. It also discusses how to use the Progress/400 Data Dictionary to maintain DB2/400 data definitions.

Chapter 10, “AS/400 Utilities”

Describes the Progress/400 server utilities that you use to create and maintain the DataServer environment on the AS/400.

Chapter 11, “Progress 4GL Interfaces to OS/400 Languages and Objects”

Documents External Programming Interfaces, executing OS/400 commands, stored procedure support, and OS/400 object access.

Appendix A, “Tutorials for Managing Your Dictionary Library”

Provides options with detailed steps to help you manage your Dictionary Library with Progress/400.

Appendix B, “Legacy Support and the Progress Unknown Value”

Discusses issues related to legacy unknown values and Progress/400 support for the SQL NULL value.

Appendix C, “Useful OS/400 Commands”

Describes useful OS/400 CL commands.

“Glossary”

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 ... ] }
       }
```

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)
```

Use Progress online help to get more information about Progress messages. On the Windows platform, 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, you can 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 guide 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.

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 commands and parameters in alphabetical order.

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 accessing non-Progress applications from Progress. This guide describes how to use system clipboards, UNIX named pipes, Windows dynamic link libraries, Windows dynamic data exchange, Windows ActiveX controls, and the Progress Host Language Call Interface to communicate with non-Progress applications and extend Progress functionality.

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 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.

Progress Database Administration Guide and Reference

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

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, for example, the *Progress DataServer for ODBC Guide* and the *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 3.1 or higher 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/prodindex/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 and deploy Java and ActiveX applications that run as clients of the Progress AppServer. The guide includes information about how to expose the AppServer as a set of Java classes or as an ActiveX server.

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 Version 3 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.

1

Introduction

The Progress/400 product allows you to access information stored in DB2/400 database files by applications written in the Progress Fourth Generation Language (4GL). You can develop your applications within the Progress Application Development Environment (ADE). The ADE includes a set of graphical tools that helps you maintain databases and develop applications. In addition to providing you with the Progress Application Development Environment (ADE) tools, the product allows you to implement Progress database features and 4GL extensions in applications that run with the DB2/400 database. Some of these tools and features are:

- **Progress/400 DataServer** — Use the Progress/400 DataServer to access DB2/400 databases and other OS/400 objects.
- **AppBuilder** — Use the AppBuilder to design and generate code for your graphical user interfaces. For example, you can use the AppBuilder to create a browse widget for viewing DB2/400 data.
- **Progress/400 Data Dictionary** — Use the Progress/400 Data Dictionary to maintain DB2/400 data definitions.
- **Native 4GL Client** — Use the Native 4GL Client to run batch and report applications on the AS/400.
- **Progress/400 AppServer** — Use the AppServer to participate in distributed application computing.

This chapter presents an overview of the Progress/400 product. It describes how Progress applications and databases work together with the DB2/400 DBMS through the DataServer. Specifically, this chapter discusses product architecture and guidelines for using the product.

1.1 Progress/400 Architecture

The Progress/400 product supports several application architectures and hardware configurations. The following section documents these architectures.

1.1.1 Client/Server, Web-based, or N-tier Architecture

The Progress/400 product allows Progress applications operating in a client/server, Web-based, or n-tier mode to access data in DB2/400 database files. The primary difference between a host-based architecture and a client/server architecture is the location of the application code. In a client/server architecture, the application code resides on the PC or workstation client and the database resides on the host AS/400. A Web-based or n-tier architecture requires the use of a middle-tier platform; however, this middle tier appears as just another client to the DataServer.

With the Progress/400 product, the Progress client (MS-Windows or UNIX) contains the Progress application logic, and requests database records from the AS/400. The DataServer retrieves database requests from the AS/400 and delivers them to the client. Each Progress client starts its own DataServer.

Figure 1–1 shows the Progress/400 client and server components.

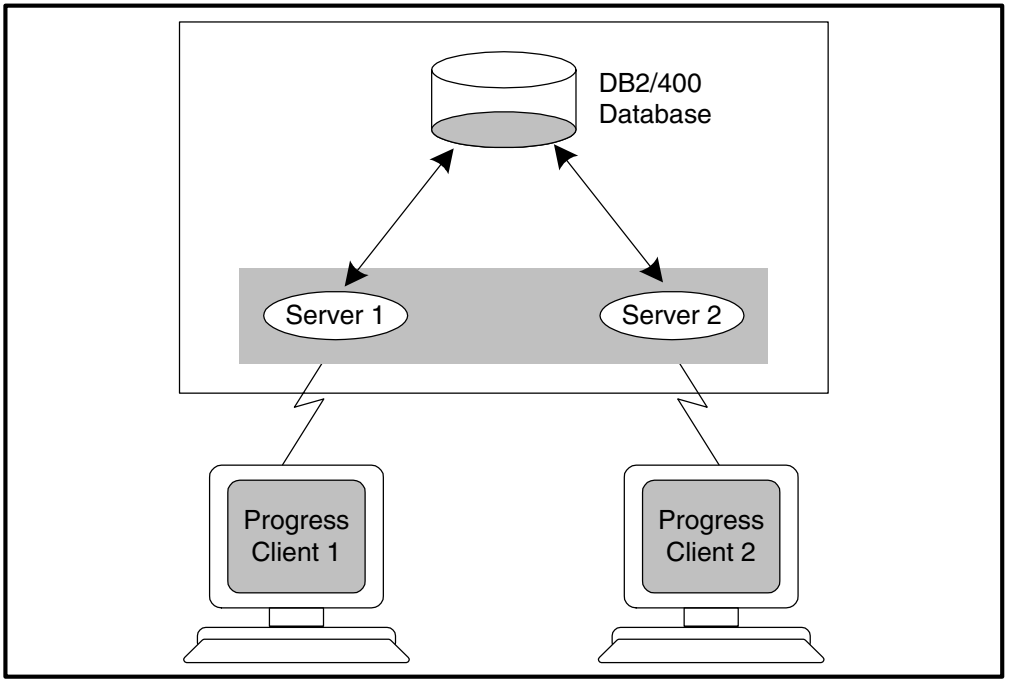


Figure 1–1: Progress/400 Client/Server Architecture

Note that in Figure 1–1:

- A connection can also be made from the middle-tier platform of a WebSpeed Transaction Server or an AppServer instead of a Progress client.
- You can connect through TCP or SNA network protocols. The SNA connection requires the use of specific routers. See the "SNA Communications Protocol" section in Chapter 4, "Remote Access to Progress/400 DataServer," for details.

1.1.2 Native 4GL and Progress/400 AppServer Architecture

The Native 4GL Client and the Progress/400 AppServer architectures make it possible for Progress 4GL code to be compiled and executed on the AS/400.

Progress applications operating on the AS/400 can access data in DB2/400 database files locally using the Native 4GL Client as well as the Progress/400 AppServer. In this configuration, both the database and the application code reside on the AS/400. This architecture can exist simultaneously with the Progress/400 client/server architecture. The Native 4GL Client is available in a run-time-only version with the Progress/400 DataServer. The Progress/400 Native 4GL Compiler product allows the Native 4GL Client to compile Progress 4GL code directly on the AS/400.

Figure 1–2 shows the Progress/400 Native 4GL Client and the Progress/400 AppServer in the host-based architecture on the AS/400.

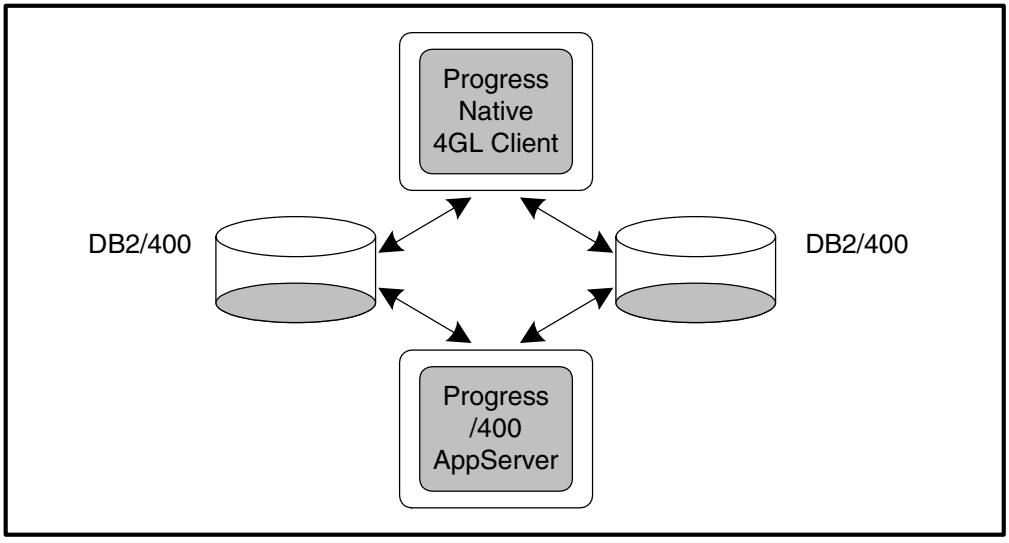


Figure 1–2: Progress/400 Host-based Architecture

With the host-based Progress/400 architecture, the Progress Native 4GL Client as well as the Progress/400 AppServer can run the Progress 4GL application on the AS/400. This takes advantage of the AS/400 server performance and reduces the network load. These clients can connect to multiple databases as long as they reside on the same AS/400. They are also compatible with all of the supported client/server configurations for accessing data. However, they do not have interactive capabilities.

1.1.3 Inside Progress/400

In a client/server architecture, accessing DB2/400 database files through the DataServer involves migrating the data definitions into the Progress/400 environment on the AS/400 and creating a schema holder on the client with those same data definitions so that Progress applications can read them. The client component also includes the Progress/400 Data Dictionary, which allows you to maintain DB2/400 data definitions from the Windows Progress client.

The central components of the Progress/400 product are the server schema, the client schema holder, and the schema image.

Figure 1–3 shows the Progress/400 internals.

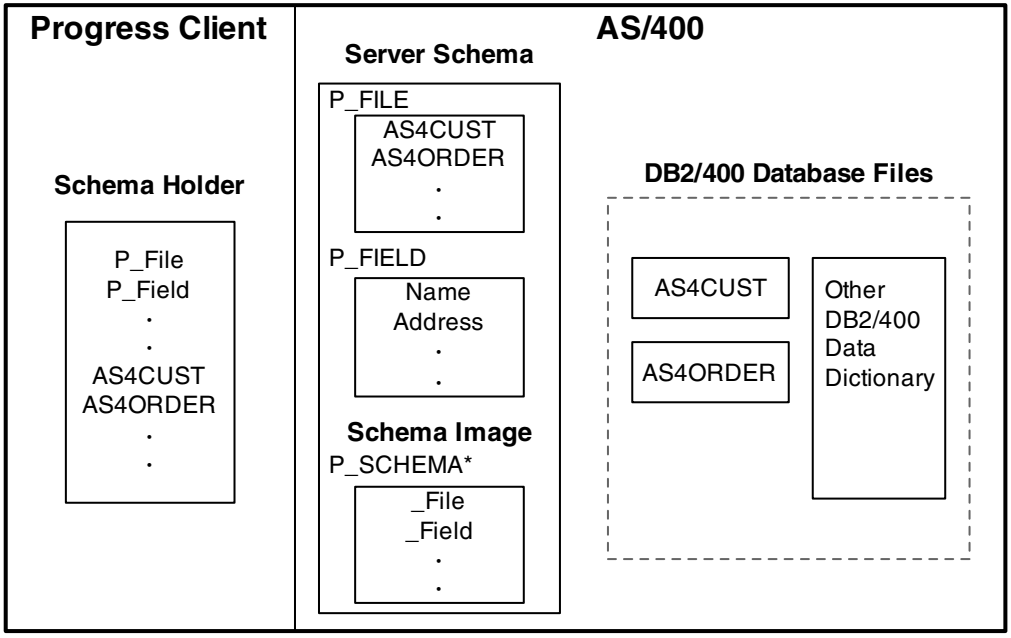


Figure 1–3: Progress/400 Internals

In the native architecture, you still migrate the data definitions into the Progress/400 environment on the AS/400, but instead of creating a schema holder remotely, you create a schema image. You can use the Windows client to synchronize your schema image.

You can manage all of your data definitions through your Windows client. However, you still have the option of maintaining the DB2/400 data definitions by means of DDS (including SQL/400 and IDDU) on the AS/400.

1.2 Progress/400 Objects

The Progress/400 product is comprised of multiple objects. This section discusses the following Progress/400 objects: server schema, Dictionary Library, schema holder, and schema image.

1.2.1 A Dictionary Library

A Dictionary Library is an AS/400 library built by Progress /400. It contains the Progress/400 Server Schema and Schema Image. In addition to the Server Schema and Schema Image, a Dictionary Library can optionally contain application data. When a Progress client connects to the AS/400, it actually connects to a Dictionary Library.

1.2.2 The Server Schema

On the AS/400, a server schema (contained in a Dictionary Library) holds the file structure and schema information that the Progress/400 DataServer uses to translate between DB2/400 definitions and Progress/400 definitions. The Dictionary Library name of the server schema is the name by which the DataServer recognizes the DB2/400 database.

You determine which DB2/400 files serve as the basis of this DB2/400 database. For example, you can include a subset of files from one library and a subset from a second library into a single server schema.

The server schema contains a series of DB2/400 physical files, each with a P__ prefix, that correspond to DB2/400 database objects. The P__ files are similar to the Progress metaschema-they have a similar organization and contain the same kind of information. Table 1–1 lists the P__files contained in the server schema.

Table 1–1: Progress/400 Server Schema Files

OS/400 File	Description
P__DB	Database definition file.
P__DDS	A source file for applying Progress/400 dictionary changes to the AS/400 server schema.
P__FIELD	Field information file.
P__FILE	File information file.
P__INDEX	Index information file.
P__IDXFD	Index field information file.
P__SCHEMA1 P__SCHEMA2 P__SCHEMA3	Data definitions used by the native clients. These three files comprise the schema image.
P__SEQ	Sequences information file.
P__TRGFD	Field trigger information file.
P__TRGFL	File trigger information file.
P__USER	Currently not used.
P__VIEW	Currently not used.
P__VCOL	Currently not used.
P__VREF	Currently not used.

For example, the sever schema represents the DB2/400 physical file AS4CUST with four fields defined-cust-num, name, address, zip-by an entry for AS4CUST in P__FILE and by four entries in P__FIELD for cust-num, name, address, and zip.

The schema image contains the data definitions for the DB2/400 database like the schema definitions stored in a schema holder for a remote client. The Native 4GL Client and the Progress/400 AppServer use the schema image to map the data definitions in the server schema to a format Progress can understand. See Chapter 5, "Preparing to Use AS/400-based Clients," for more information on creating the schema image.

1.2.3 Progress/400 Dictionary Objects

Any object that Progress/400 creates in a dictionary library (other than database files) must remain unchanged. Progress/400 relies on the knowledge that certain objects will always be in a dictionary library when dictionary maintenance of any type is performed or when a connection is made through a Progress/400 server. This applies to the following objects:

- Schema files
- User spaces
- Journals and receivers
- User indexes

1.2.4 The Schema Holder

A *schema holder* is a standard Progress database residing on the remote client that contains schema definitions for one or more non-Progress databases. The *schema* of a database is a description of its structure, the files, the fields within the files, the indexes, and so forth. The Progress/400 DataServer uses the schema holder to map the data definitions in the server schema to a format that Progress can understand.

Before a Progress client can access DB2/400 data, you must create a schema holder. The process is slightly different depending on whether you are accessing existing DB2/400 database files or moving a Progress database to the AS/400. See Chapter 3, "Creating the Progress/400 Environment," for information on building a schema holder. The process is similar for both scenarios, but there are minor differences in the way the DataServer loads schema information into the client schema holder.

1.2.5 The Schema Image

A *schema image* contains schema definitions for a DB2/400 database. The schema is a description of a database structure, the files, the fields within the files, the indexes, and so forth. The Native 4GL Client and the Progress/400 AppServer use the schema image to understand the rest of the server schema.

Before the Native 4GL Client or the Progress/400 AppServer can access DB2/400 data, you must create a schema image. See Chapter 5, "Preparing to Use AS/400-based Clients," for information on building a schema image.

1.2.6 Demonstration Databases

Progress/400 provides DB2/400 databases that contain the tables, fields, indexes, and data found in the standard Progress sports, sports2000, and demo databases. On the AS/400, these databases are called PROSPORT, SPORTS2000, and PRODEMO, respectively. Progress Software Corporation supplies them on the same media as the other Progress/400 software. You can create a copy of these databases to practice working with the Progress language or the Progress/400 utilities. These are the general steps that you follow to create a copy of a demonstration database:

1. Use the Duplicate Progress/400 Database (DUPPRODB) server utility to copy the demonstration database. The utility creates a server schema with the data definition information and data from the specified database files.

2. Specify *PROSPORT, *SPORTS2K, or *PRODEMO for the From Progress/400 Database Name (FROMDB) parameter.

3. Specify Full Copy *YES.

4. On the client machine, create a schema holder.

See Chapter 3, “Creating the Progress/400 Environment,” for specific instructions on how to run DUPPRODB and how to create a client schema holder.

1.3 Progress/400 Utilities

The client and server utilities allow you to manage the Progress/400 environment.

The Windows client has the Progress/400 Data Dictionary tool. This tool allows you to create and modify DB2/400 data definitions within the Progress visual Application Development Environment (ADE). The Progress/400 Data Dictionary tool also includes dump and load utilities that allow you to load directly to the AS/400. Additionally, you can use the Progress/400 Data Dictionary to synchronize a schema image.

The client also provides a set of DataServer utilities available from the standard Progress Data Administration tool and the character client Data Dictionary for managing the remote client schema holder.

The native utilities allow you to do the following tasks:

- Manage server schema, dictionary libraries, and schema images
- Upgrade from prior versions
- Manage connectivity and Progress/400 jobs

See Chapter 9, "Remote Client DB2/400 Utilities," and Chapter 10, "AS/400 Utilities," for detailed descriptions of these utilities.

1.4 Using Progress/400

The Progress/400 product set includes:

- Progress/400 DataServer and Runtime Native 4GL Client
- Progress/400 AppServer
- Native 4GL Compiler

How you use the product depends on whether you plan to access information in existing DB2/400 database files through a Progress application, or whether you plan to migrate a Progress database to the AS/400. It also depends on whether you have used an earlier version of the product and want to upgrade your applications. Additionally, you might want to run Progress applications locally on the AS/400. The following sections briefly outline these possibilities and point you to the information you need.

1.4.1 Accessing Existing DB2/400 Database Files

These are the general steps for setting up the DataServer components so that you can access DB2/400 database files through Progress applications:

1. Install Progress on the AS/400 and on the client machine.
2. Establish basic connectivity between the client and the AS/400.
3. Create a Progress/400 server schema on the AS/400 that contains data definition information from the DB2/400 database files that you want to access.
4. Create a schema holder on the client machine.

1.4.2 Migrating Progress Databases to the AS/400

These are the general steps for setting up the DataServer components so that you can migrate a Progress database to the AS/400:

1. Determine whether you must modify any application code to handle differences between accessing Progress and DB2/400 databases.
2. Install Progress on the AS/400 and on the client machine.

3. Establish basic connectivity between the client and the AS/400.
4. Dump the data definitions and, optionally, data from the Progress database.
5. Create an empty Progress/400 server schema on the AS/400.
6. Create a schema holder on the client machine.
7. Load data definitions to the server schema on the AS/400 with the Progress/400 Data Dictionary tool.
8. Synchronize the client schema holder with the server schema.
9. Optionally, load data.

1.4.3 Upgrading to the Current Progress/400 Product

There are slight differences between upgrading from different earlier versions; however, these are the general steps to follow:

1. Install Progress on the AS/400 and on the client machine.
2. Establish basic connectivity between the client and the AS/400.
3. Create an empty Progress/400 server schema on the AS/400.
4. If converting from a version earlier than Version 9.0A, convert the earlier version server schema to the current version by running the CVTPRODCT server utility.
5. Create a schema holder on the client machine.

1.4.4 Running Progress Applications on the AS/400

Follow these general steps to run your Progress applications on the AS/400:

1. Install Progress/400 on the AS/400 and Progress on the client machine.
2. Create a server schema library that contains a schema image.
3. Transfer your Progress applications to the AS/400.

1.4.5 Security

In the DataServer architecture, there are three levels at which you should consider security issues:

- **Schema-holder security** — Implement standard Progress security for the schema holder on the client as needed. For more information on schema-holder security, see the chapter on security in the *Progress Database Administration Guide and Reference*.
- **Application security** — Implement security at the application level. For more information on application-level security, see Chapter 2, "Common Product Information," in this manual and the chapter on security in the *Progress Programming Handbook*.
- **OS/400 security** — Implement standard OS/400 security for the DB2/400 database files. Progress applications accessing DB2/400 files are subject to any security for these files. See Chapter 8, "System Administration," for more information on database security.

1.5 Where to Find Further Information

The following section documents where to find information on the Progress/400 products within this manual and across the Progress documentation set.

1.5.1 This Manual

Table 1–2 suggests paths through this guide that accommodate different approaches to using the Progress/400 product.

Table 1–2: How to Use This Manual

If You Are . . .	Read This . . .
Accessing existing DB2/400 database files	Chapter 2, “Common Product Information” Chapter 3, “Creating the Progress/400 Environment” Chapter 4, “Remote Access to Progress/400 DataServer” Chapter 9, “Remote Client DB2/400 Utilities” Appendix A, “Tutorials for Managing Your Dictionary Library”
Migrating a Progress database to the AS/400	Chapter 2, “Common Product Information” Chapter 3, “Creating the Progress/400 Environment” Chapter 4, “Remote Access to Progress/400 DataServer” Chapter 9, “Remote Client DB2/400 Utilities” Appendix A, “Tutorials for Managing Your Dictionary Library”
Running Progress applications on the AS/400	Chapter 2, “Common Product Information” Chapter 3, “Creating the Progress/400 Environment” Chapter 4, “Remote Access to Progress/400 DataServer” Chapter 5, “Preparing to Use AS/400-based Clients” Chapter 6, “Using the Progress/400 Native 4GL Client” Chapter 7, “Using the Progress/400 AppServer” Chapter 9, “Remote Client DB2/400 Utilities” Chapter 10, “AS/400 Utilities”
Utilizing objects and high-level languages on the AS/400	Chapter 11, “Progress 4GL Interfaces to OS/400 Languages and Objects”

1.5.2 Related Topics

This guide includes information on the concepts, procedures, and commands related to using the Progress/400 product set. However, if the text mentions topics not found in this guide, it refers you to the appropriate documentation for details. For example, if you read a section that describes writing Progress code, the text might refer you to the *Progress Language Reference* or the *Progress Programming Handbook* for more details.

If you have never used Progress and need an introduction, start with the *Progress Application Development Environment — Getting Started* and *Progress Language Tutorial for Windows* or *Progress Language Tutorial for Character* manuals.

Table 1–3 presents a list of topics related to developing and running applications for the Progress/400 DataServer and indicates where to find the documented information.

Table 1–3: Progress-related Topics *(1 of 2)*

Topic	Documentation
Starting Progress	*Progress Installation and Configuration Guide Version 9 for Windows* *Progress Installation and Configuration Guide Version 9 for UNIX*
Setting up the AppServer	*Progress Installation and Configuration Guide Version 9 for UNIX*
Using the AppServer	*Building Distributed Applications Using the Progress AppServer*
Creating or copying a Progress database using the `prodb` command	*Progress Database Administration Guide and Reference*
Defining security for a Progress database	*Progress Database Administration Guide and Reference*
Using the Progress Data Dictionary	*Progress Language Tutorial for Windows* or *Progress Language Tutorial for Character* *Progress Basic Development Tools* (Character only; information for Windows is in online help)
Using the Progress Application Development Environment	*Progress Application Development Environment — Getting Started*

Table 1–3: Progress-related Topics *(2 of 2)*

Topic	Documentation
Writing applications in the 4GL	*Progress Language Tutorial for Windows* or *Progress Language Tutorial for Character* *Progress Language Reference* *Progress Programming Handbook*
Compiling 4GL applications	*Progress Language Tutorial for Windows* or *Progress Language Tutorial for Character* *Progress Language Reference*
Upgrading to Progress Version 9	*Progress Product Update Bulletin*

2

Common Product Information

The components of the Progress/400 product have common features and attributes. This chapter discusses these features and attributes in the context of:

- Database design issues
- Data types
- Transactions
- Queries
- Field lists
- Language restrictions
- Application security
- Progress/400 word index support

2.1 Database Design Issues

The Progress/400 product allows an application to access data stored in DB2/400 database files. When you create or modify the Progress or DB2/400 databases that your applications access, you might have to consider issues such as Progress and DB2/400 database objects, naming conventions, indexes, and views. The following sections discuss the differences between Progress and DB2/400 in these areas and explain how the DataServer deals with them.

2.1.1 DB2/400 and Progress Objects and Terminology

DB2/400 and Progress databases share some of the elements common to data storage systems, but each has additional elements. For DB2/400, you define these elements in physical and logical files. For a Progress database, you define objects with the Data Dictionary. In addition, the two database systems use different terminology to refer to these objects. Table 2–1 compares the DB2/400 and Progress database objects.

Table 2–1: Progress and DB2/400 Objects *(1 of 2)*

Progress Object	DB2/400 Equivalent
Table	File
Field	Field
Record	Record
Index	File key and logical file
Sequence	Sequence
Trigger	Trigger
Validation expression	Validity checking
Validation message	No equivalent
View	Joined logical file
No equivalent (Progress/400 supports it as a virtual table)	Limited logical file
No equivalent (Progress/400 supports it as a virtual table)	Multi-record logical file

Table 2–1: Progress and DB2/400 Objects *(2 of 2)*

Progress Object	DB2/400 Equivalent
No equivalent (Progress/400 supports it as a virtual table)	Program-described logical file
No equivalent (Progress/400 supports it as a virtual table)	Multi-record program-described logical file

2.1.2 Naming Conventions

One factor to consider when designing for consistency between Progress and DB2/400 is the kind of restrictions each has for naming tables or files and fields. Progress/400 handles the conflicts in naming conventions for you.

Object Names

These are the restrictions for naming Progress and DB2/400 database objects, including files, tables, fields, and indexes:

- **Progress** — Names can be up to 32 characters long and can consist of alphabetic characters (A-Z and a-z), digits (0-9), and the special characters $, &, #, %, -, and _. Names must begin with an alphabetic character and are not case sensitive.
- **DB2/400** — Names can be up to 10 characters long and must follow OS/400 naming conventions. Progress/400 supports SQL long names as input to the server schema through the CHGPRODCT utility. For a description, see Chapter 10, "AS/400 Utilities."

NOTE: The prefix, P__, is reserved for files in the Progress/400 server schema.

The set of characters that Progress allows includes all of the characters DB2/400 allows, so no modifications to the DB2/400 file or field names need to be done before a Progress application can reference them. However, when you move a Progress database to the AS/400, Progress/400 modifies Progress names as necessary according to these rules:

- Names appear in uppercase.
- Names longer than 10 characters are truncated.
- If the first character is not A-Z, a-z, #, $, or @, Progress/400 replaces it with an "A."
- If subsequent characters are not A-Z, a-z, 0-9, #, $, @, period (.), or comma (,) they are replaced with an underscore (_).

For example, the property sheet for a field named sales-representative shows the AS/400 name to be SALES_REPR.

Unique Table and File Names

Progress requires that a table name be unique within a database. DB2/400 requires that a filename be unique within a library. Because you can include database files from multiple libraries in a single Progress/400 dictionary library, you might have duplicate names.

Progress/400 resolves nonunique table names for you when creating or modifying the server schema. For example, it might remove underscores and embedded vowels, and append underscores (_). The properties sheet in the Progress/400 Data Dictionary shows you which DB2/400 file mapped to a table in the schema. When you create files using the Progress/400 Data Dictionary, you must enter unique Progress names.

2.1.3 Indexes and Keys

Progress indexes and DB2/400 file keys behave the same way. Queries that you develop for a Progress database work for DB2/400 database files.

If a DB2/400 file has more than one index, the first index processed is the primary index for that file. The first index can be the keyed physical file or a logical file. If a DB2/400 file has no index, Progress/400 processes records in relative record order. When creating indexes with the Progress/400 Data Dictionary, the first index created defaults to the physical file key and the primary index.

Progress allows a maximum of 16 fields when defining an index. The Progress length of the combined fields that make up an index must be less than 200 bytes. The Progress/400 maximum length is some number less than 200, depending on the number of key fields used.

Case-insensitive Indexes

By default, Progress is case insensitive, but you can define a field as case sensitive. DB2/400 is case sensitive, but the Progress/400 product provides support for case insensitivity.

To enable DB2/400 database files to support case insensitivity, you must specify a case-insensitive Alternate Sequence Collating Table when you run DUPPRODB on the AS/400. See Chapter 8, "System Administration," for more information on user-defined collation tables.

If at least one index field is case sensitive, the entire index is case sensitive.

If all index fields are case insensitive and you specified a code-page value for the DUPPRODB ALTSEQCI keyword, that code-page value is specified as the DDS ALTSEQ parameter when the logical file is compiled. Otherwise, the ALTSEQCI keyword value of the `INSTALL` command is used.

ASCII Versus EBCDIC Sort Order

The AS/400 uses the EBCDIC coding scheme, which uses a different sort order (collation sequence) than ASCII. For example, an index sorted according to ASCII values returns records in the order of 0 to 9, then A to Z, followed by a to z. An index sorted according to EBCDIC values returns a to z, then A to Z, followed by 0 to 9. Other characteristics of the EBCDIC coding scheme are that alphabetic characters are not contiguous and special characters are in a different order than in ASCII.

If you are moving a Progress database to the AS/400 and want to maintain an ASCII sort order for your indexes, specify an ALTSEQ table with an ASCII collation sequence when you run the Duplicate Progress Dictionary and DB2/400 Database (DUPPRODB) utility on the AS/400. See Chapter 8, "System Administration," for more information on ALTSEQ tables.

If your application relies on an ASCII sort order when it accesses DB2/400 database files, create a logical file on the AS/400 that includes the appropriate ALTSEQ table so that the index sorts in ASCII collation sequence.

2.1.4 Triggers

Database triggers are code that an application associates with a database object and an action. For example, writing a record can cause code associated with that object or action to execute. DB2/400 supports insert before and after, update before and after, and delete before and after triggers. Your database files can contain DB2/400 triggers and Progress triggers that you added using the Progress/400 Data Dictionary tool.

2.1.5 Views

There is no direct equivalent for Progress views in DB2/400. However, DB2/400 database files can include a type of logical file that provides the same functionality as a view-the joined logical file. The Progress/400 Data Dictionary lists joined logical files as tables. You cannot update data accessed with joined logical files.

2.1.6 Virtual Tables

The DataServer can access both physical and logical files on the AS/400. Physical files are analogous to Progress tables and logical files are analogous to Progress indexes. The AS/400 supports three distinct types of logical files in addition to those that support standard indexes:

- **Joined logical files** — Joins two or more logical files
- **Limited logical files** — Includes a subset of fields from a physical file
- **Multiple record logical files** — Associates two or more physical files

To support these types of logical files, the DataServer places limitations on how and where you can use them. A logical file is classified as a Progress/400 virtual table if it is a join, if its record format is different from the physical file on which it is based, or if it contains more than one record format.

In the case of multiple record logical files, the DataServer treats each record format as a separate table. For example, the logical file ORDERLIN has two record formats identified as ORDER and ORDER_LIN. The DataServer represents this file as two tables in the Progress/400 Data Dictionary-ORDERLIN_ORDERR and ORDERLIN_ORDER_LINR.

You can view virtual tables through the Progress/400 Data Dictionary. You cannot maintain them because the Progress/400 Data Dictionary has read-only access to virtual tables.

Virtual tables do not support record positioning by relative record number. You cannot use RECID to access records in a virtual table. Any use of RECID with virtual tables produces invalid results. The following code example uses RECID with virtual tables and therefore does not return a valid record:

```
DEFINE VAR save-recid AS RECID.
FOR EACH VIRTFILE NO-LOCK:
  save-recid = RECID(VIRTFILE).
  FIND FIRST VIRTFILE WHERE RECID(VIRTFILE) = save-recid.
END.
```

For information on relative record numbers, see your DB2/400 documentation.

2.1.7 Multiple Members in Physical Files

Progress/400 does not support logical files that are built over multiple members in a physical file. Suppose, for example, that the physical file SAMPLE contains the members M1, M2, M3, and M4. You now create the logical file SAMPLEL from SAMPLE and then enter the `ADDLFM` (Add Logical File Member) command as follows:

```
ADDLFM FILE(SAMPLEL) MBR(SAMPLEL) DTAMBRS(*CURRENT/SAMPLE(M1 M2 M3 M4))
```

This command adds the member SAMPLEL, built over all of the members included in the DTAMBRS parameter, to the logical file SAMPLEL.

Progress/400 uses relative record numbers for its internal processing and positioning. The DataServer cannot position to a record in a structure such as this because ILE/C does not directly support it. As a work around, simply create matching members in both the logical and physical files. Use the following OS/400 commands to add the logical file members:

```
ADDLFM FILE(SAMPLEL) MBR(M1) DTAMBRS((*CURRENT/SAMPLE (M1))
ADDLFM FILE(SAMPLEL) MBR(M2) DTAMBRS((*CURRENT/SAMPLE (M2))
ADDLFM FILE(SAMPLEL) MBR(M3) DTAMBRS((*CURRENT/SAMPLE (M3))
ADDLFM FILE(SAMPLEL) MBR(M4) DTAMBRS((*CURRENT/SAMPLE (M4))
DO TRANSACTION:
   CREATE QCMD.
   ASSIGN cmd = "!CLOSE SAMPLE".
   RELEASE QCMD.
   CREATE QCMD.
   ASSIGN cmd = "! OVRDBF FILE(SAMPLEL) TOMBR(M4)".
   RELEASE QCMD.
   CREATE QCMD.
   ASSIGN cmd = "! OVRDBF FILE(SAMPLE) TOMBR(M4)".
   RELEASE QCMD.
END.
```

The result is that the logical file SAMPLEL now contains the members M1, M2, M3, and M4.

The following code can successfully retrieve the correct record from member M4:

```
...
FOR EACH sample USE-INDEX SAMPLEL
...
```

2.1.8 Progress/400 and Distributed Data Management

Distributed Data Management (DDM) is an AS/400 facility that allows DB2/400 files from one AS/400 to be accessed by other AS/400s. The High Level Language programs RPG, COBOL, C, as well as 4GL programs written by the Progress 4GL can access these DDM files. Progress/400 DataServer supports DDM files that reference either physical or logical files. The following section documents how to access DDM files from the Progress/400 DataServer and from the Progress/400 native clients.

DDM Files

DDM files are like pointer objects. They point to actual physical or logical files on a remote AS/400. DDM file access uses SNA over an APPC network. It is possible to use TCP in place of SNA.

Figure 2–1 shows the DDM architecture.

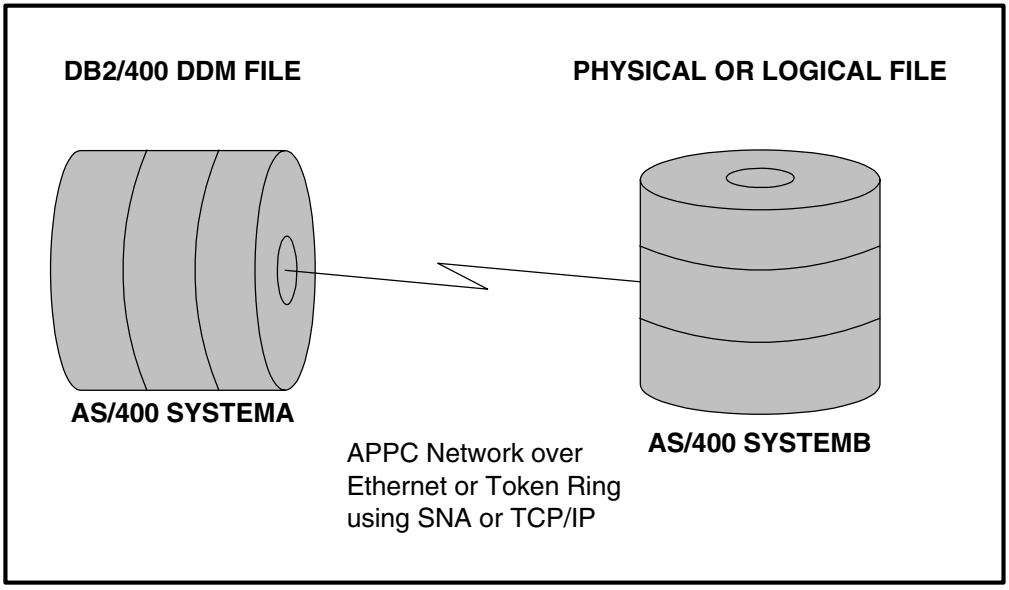


Figure 2–1: DDM Architecture

In Figure 2–1, assume that SYSTEMB has one physical file, CUSTOMER, and one logical file or index, NAME, in the library SPORTS. In this example suppose programs executing on SYSTEMA need access to these files for some function. For the simplest access, create the DDM files on SYSTEMA into the library SPORTS. To do this, execute the following commands on SYSTEMA:

```
CRTLIB LIB(SPORTS)

CRTDDMF FILE(SPORTS/CUSTOMER) RMTFILE(SPORTS/CUSTOMER) RMTLOCNAME(SYSTEMB)

CRTDDMF FILE(SPORTS/NAME) RMTFILE(SPORTS/NAME) RMTLOCNAME(SYSTEMB)
```

After you execute these commands, library SPORTS on SYSTEMA contains 2 DDM files, named exactly the same as the physical and logical files on SYSTEMB. When the DDM files are opened on SYSTEMA, the data is retrieved from SYSTEMB. This is the simplest DDM implementation. You also can add Progress/400 to this design.

Progress/400 and DDM Files

For the purpose of adding Progress/400 to the design in Figure 2–1, assume that SYSTEMB has a library of files that must be accessed from SYSTEMA using the Progress/400 DataServer.

Assume these files reside in the library SPORTS. In order to allow Progress/400 access to these files, you must pull their definitions into Progress/400 using the Progress/400 command CHGPRODCT. This command pulls the definitions of DB2/400 files into an existing Progress/400 Server Schema. To allow Progress/400 to assess the files in the library SPORTS, use the following commands on SYSTEMB:

```
ADDLIBLE PROGRESS *LAST

DUPPRODB NEWDB(SPORTSDL) FRMDB(*PROEMPTY)
        /* this command creates a Progress/400 Dictionary Library
        called SPORTSDL. */

CHGPRODCT PRODCT(SPORTSDL) FRMFILLST((*ALL SPORTS))
        /* this command pulls the file definitions of the files
        in SPORTS into the Server Schema in library SPORTSDL. */
```

Once you have executed these commands and created a schema holder, Progress 4GL programs can now access the files on SYSTEMB. However, the Progress/400 DataServer must be executing on SYSTEMB first. In order to gain access to the files in library SPORTS from SYSTEMA, follow these steps:

1 ♦ On SYSTEMB, save the SPORTSDL library to tape or a save file using the SAVLIB command.

2 ♦ On SYSTEMA:

a. Restore the library SPORTSDL using the RSTLIB command.

b. Execute the following command:

```
CRTLIB SPORTS
```

c. For each physical and logical file in the library SPORTS on SYSTEMB, execute the following:

```
CRTDDMF FILE(SPORTS/filename) RMTFILE(SPORTS/filename)
          RMTLOCNAME(SYSTEMB)
CRTDDMF FILE(SPORTS/filename1) RMTFILE(SPORTS/filename1)
          RMTLOCNAME(SYSTEMB)
```

Continue until you have created all the DDM files.

Once you have successfully created the DDM files, the Progress/400 DataServer can access the Progress/400 dictionary library SPORTSDL on SYSTEMA. Now that the SPORTSDL library exists on SYSTEMA, you must create a client schema holder so that the Progress 4GL can access it. Once this is complete, connect to SPORTSDL using the CONNECT statement with the `-is` parameter added to the connection parameters. This parameter prevents Level Checking, which cannot be used when accessing DDM files. The following commands show a sample connection with the additional `-is` parameter:

```
CONNECT SPORTSH -1.
CONNECT SPORTDL -H SYSTEMA -H USER -P PASSWD -N TCP -S service -is.
```

2.2 Data Types

The following sections describe how the DataServer maps data types. Most Progress data types map directly to DB2/400 data types; the few exceptions are noted. Field property sheets in the Progress/400 Data Dictionary display the Progress and DB2/400 data types.

2.2.1 DB2/400-to-Progress Data Type Mapping

Table 2–2 shows how the DataServer maps data types in existing DB2/400 database files to Progress data types when you create a client schema holder. For each DB2/400 data type, the table lists the DDS equivalent, the Progress/400 label, and the DataServer implementation.

Table 2–2: DB2/400-to-Progress Data Type Mapping *(1 of 2)*

DB2/400 Type	DDS Type	Label	Progress Equivalent
Character	A	Character	CHARACTER
Character	A	Case-insensitive key string[1]	CHARACTER
Character	A	Logical	LOGICAL
Zoned decimal	S	Zoned numeric	DECIMAL
Packed decimal	P	Packed decimal[2] Packed decimal (even digits)	DECIMAL
Binary	B	Signed 2 byte[3] Signed 4 byte	INTEGER

Table 2–2: DB2/400-to-Progress Data Type Mapping *(2 of 2)*

DB2/400 Type	DDS Type	Label	Progress Equivalent
Floating Point	F	Short floating pt Long floating pt	DECIMAL
Date[4]	L	Date	DATE
Time[5]	T	Time	CHARACTER
Timestamp[5]	Z	Timestamp	CHARACTER

1 Using the case-insensitive character string data type prevents the DataServer from performing selection by server.

2 The packed decimal data type is labeled packed decimal (odd) or packed decimal (even digits) depending on whether the field contains an odd or even number of digits, respectively.

3 The size of a signed 2-byte field cannot exceed 4 digits.

4 Progress supports two display formats for DATE: mm/dd/yy and mm/dd/yyyy. Use the Progress/400 Data Dictionary to specify a display format. If you want to display dates in other formats, you can use the Progress Date Format (-d) startup parameter or you can manipulate the data through your application code.

5 Progress does not have comparable data types. When you access a DB2/400 database, the Progress client ensures that only valid values can be inserted in fields of this type. However, if you dump the definitions from a DB2/400 database and load the definition to a standard Progress database (non-schema holder), these data types are implemented as character fields and lose the error checking.

2.2.2 Progress-to-DB2/400 Data Type Mapping

The following sections describe how the DataServer converts data types when you port an existing Progress database to DB2/400.

CHARACTER

Progress CHARACTER fields map to DB2/400 single-byte character set (SBCS) fields, which correspond to the DDS A data type. The display format specified in the Progress Data Dictionary determines the DB2/400 field size. For example, the Customer.Name field has a display format of 20 characters. The equivalent DB2/400 field length is 20 bytes.

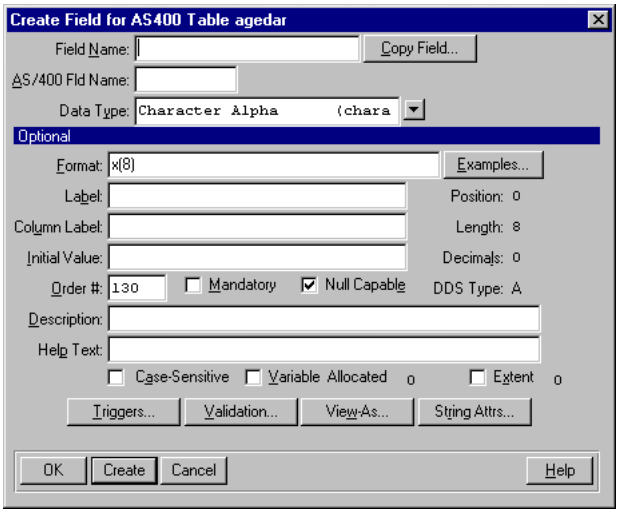
The DB2/400 database can store character data in variable-length character strings. To enable this function when creating character fields in the Progress/400 Data Dictionary, follow these steps:

1 ♦ Choose the Field mode button in the Progress/400 Data Dictionary main window.

2 ♦ Choose Modify schema.

3 ♦ Select the table that will contain new fields from the Tables list.

4 ♦ Choose the Create Field button. The following dialog box appears:



5 ♦ Enter a Format mask equal to the total maximum length.

6 ♦ Select the Variable Allocated toggle box and enter the allocated amount.

If you enter a Format of x(500) and allocate 50, DB2/400 will store the first 50 characters in the allocated area of the field with a pointer to any remaining characters. Additionally, the amount you allocate must be less than the maximum length of the Format. Choose the Help button to access detailed information on all of the elements in the dialog box.

7 ♦ Choose OK.

DATE

Progress DATE fields map to DB2/400 date fields, which correspond to the DDS L data type. The DataServer creates a DB2/400 date field whose internal storage format is *ISO, which stores dates as yyyy-mm-dd. Table 2–3 lists the date formats supported by DB2/400.

Table 2–3: DB2/400 Date Formats

DB2/400 Date Type	Description and DB2/400 Storage Format
*EUR	IBM European Standard. Storage format = dd.mm.yyyy.
*ISO	International Standards Organization. Storage format = yyyy-mm-dd.
*JIS	Japanese Industrial Standard Christian Era. Storage format = yyyy-mm-dd.
*USA	IBM USA Standard. Storage format = mm/dd/yyyy.
*DMY	Day/Month/Year. Storage format = dd/mm/yy.
*MDY	Month/Day/Year. Storage format = mm/dd/yy.
*YMD	Year/Month/Day. Storage format = yy/mm/dd.
*JUL	Julian. Storage format = yy/ddd.

When a Progress application accesses dates in a DB2/400 database, it displays them in one of these formats, regardless of the internal storage format:

- mm/dd/yy
- mm/dd/yyyy

Use the Field Properties sheet in the Progress/400 Data Dictionary tool to switch between formats. To display dates in other formats, use the Date Format (-d) startup parameter or manipulate the data through your application code.

DECIMAL

Progress DECIMAL fields map to DB2/400 packed decimal numbers, which correspond to the DDS P data type. Depending on the number of digits in the DECIMAL field, the data type maps to packed decimal (odd digits) or to packed decimal (even digits).

DB2/400 requires that you define precision (number of digits) and scale (number of digits to the right of the decimal point). The DataServer uses the value of the Decimals field in the definition as the value for the scale. It uses the number of digits in the format field as the value for precision. If no format information is available, the DataServer uses the Progress default format (->>,>>9.99) as the value for precision.

INTEGER

Progress INTEGER fields map to DB2/400 integers, which correspond to the DDS B data type. Whether an INTEGER field maps to a four-byte or a two-byte integers depends on the size of the display format:

- If the display format has four digits or less, the field becomes a two-byte integer.
- If the display format has more than four digits, the field becomes a four-byte integer.

LOGICAL

Progress LOGICAL fields map to single-character fields in DB2/400. A 0 represents false, a 1 represents true, and a question mark (?) represents the unknown value. The Progress/400 Data Dictionary labels these fields as Logical.

2.2.3 Progress Unknown Value Support

Progress/400 uses SQL NULL support to provide the Progress unknown value. The DB2/400 field must allow nulls to support this feature.

When you create a field using the Progress/400 Data Dictionary, you can select Null Capable, which allows the field to have no initial value. If you assign a question mark (?) in the Initial Value field of the field Property sheet, no initial value is assigned. If you want to assign a value to a NULL capable field, use the ASSIGN or UPDATE statements and enter data.

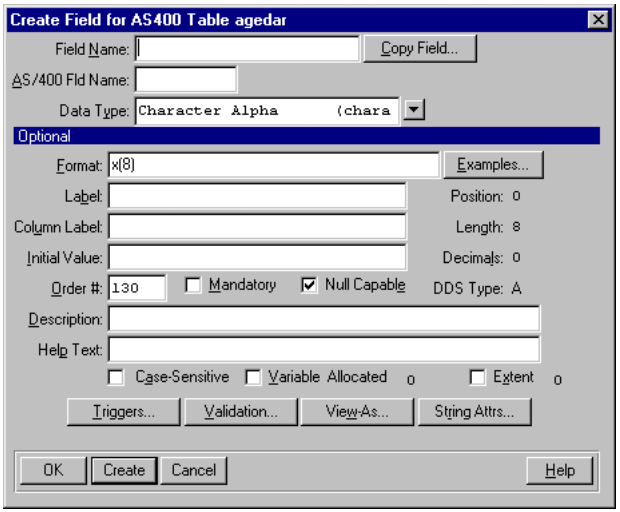
Follow these steps to select the NULL Capable option for a field:

1 ♦ Choose the Field mode button in the Progress/400 Data Dictionary main window.

2 ♦ Choose Modify Schema.

3 ♦ Select the table that will contain the new field from the Tables list.

4 ♦ Choose the Create Field button. The following dialog box appears:



You can select the Mandatory or the Null Capable option setting. Table 2–4 displays possible settings for these two values. This table also describes which assignments are allowed based on these option settings.

Table 2–4: Progress/400 Data Dictionary Options for SQL NULL and Unknown Value Assignments

Mandatory	NULL Capable	Result
ON	ON	4GL prevents (?) assignment for unknown value.
OFF	ON	Any value is legal.
ON	OFF	4GL prevents (?) assignment.
OFF	OFF	Any value is legal.

5 ♦ Enter a question mark (?) in the Initial Value field.

6 ♦ Activate the Null Capable option.

NOTE: If you currently have data files that do not support SQL NULLs and have the legacy unknown value, see Appendix B, "Legacy Support and the Progress Unknown Value."

2.3 Transactions

The Progress RDBMS and the Progress/400 DataServer process records differently before a transaction completes or is committed. The differences are as follows:

- In the Progress 4GL, transactions end at the end of the outermost block where an update takes place. When a transaction that updates a DB2/400 database ends successfully, the Progress/400 DataServer sends a COMMIT message to the DB2/400 data manager. If the transaction is interrupted, Progress sends a ROLLBACK message to the DB2/400 RDBMS.
- If the transaction deletes records, the Progress RDBMS views the records as locked until the transaction is complete **except** when another user attempts to read the record with NO-LOCK, in which case Progress views the record as deleted.
- If the transaction deletes records, Progress/400 views the records as deleted **except** when another user attempts to insert the record, in which case Progress/400 views them as locked.

This difference is illustrated in Table 2–5, where a Progress transaction deletes RECORD A before the transaction is committed. In this example, RECORD A means a record with a specific key, such as CUSTOMER WHERE CUST-NUM = 300.

Table 2–5: Standard Progress and Progress/400 Transactions *(1 of 2)*

User 2 Activity	Standard Progress			Progress/400		
	Available	Locked	View	Available	Locked	View
Inserts Record A	No	Yes	Locked	No	Yes	Locked
Reads Record A, SHARE-LOCK	No	Yes	Locked	No	No	Deleted

Table 2–5: Standard Progress and Progress/400 Transactions *(2 of 2)*

User 2 Activity	Standard Progress			Progress/400		
	Available	Locked	View	Available	Locked	View
Reads Record A, EXCLU-SIVE-LOCK	No	Yes	Locked	No	No	Deleted
Reads Record A, NO-LOCK	No	No	Deleted	No	No	Deleted

2.3.1 Record Creation

Records are created differently when you use the Progress 4GL against DB2/400 database files rather than against a standard Progress database.

The Progress RDBMS attempts to create a record when the fields that make up the index are supplied. If the entry is a duplicate key, the duplicate key error is detected immediately and the record creation fails.

The Progress/400 DataServer attempts to create a record at the end of a transaction block. If the entry is a duplicate key, the duplicate key error is detected and the record creation fails. To make your DataServer applications behave more like a Progress database, use the RELEASE or VALIDATE statements once all key field values are provided.

The following code samples illustrate how differences in record creation might affect your applications and shows how to code around them:

```
DEFINE BUFFER xcust FOR cust.
CREATE cust.
cust-num = 111.
FIND xcust WHERE xcust.cust-num = 111.
DISPLAY xcust.
```

In this example:

- Standard Progress displays customer 111.
- The Progress/400 DataServer fails to find customer 111 because it has not yet written customer 111 to the database.

To obtain standard Progress behavior against a DB2/400 database, use VALIDATE or RELEASE, as shown in this code:

```
DEFINE BUFFER xcust FOR cust.
CREATE cust.
cust-num = 111.
VALIDATE cust. /* or RELEASE cust. */
FIND xcust WHERE xcust.cust-num = 111.
DISPLAY xcust.
```

Using the RELEASE statement might not be appropriate for all applications. When you release the record, you lose your lock on it and run the risk of another user locking the record before you are done with it. Also, if a release statement is issued in a subtransaction, the lock is not released until the end of the main transaction.

2.3.2 Record Locking

The Progress RDBMS supports three lock states at the record level: EXCLUSIVE-LOCK, SHARE-LOCK, and NO-LOCK. The Progress/400 DataServer supports either these three lock states or, if you choose, the standard DB2/400 lock states LOCK and NO-LOCK. This section describes how to implement either form of record locking at the dictionary library level.

Standard Progress Locks

To support the standard Progress locking technique, the Progress/400 DataServer accesses a Progress lock table object on the AS/400. If the lock table is not present, Progress locking behavior is not supported. The lock table locks records for the physical files that are defined as part of the Progress/400 server schema. It ensures that DB2/400 records accessed by Progress applications are locked in the same manner as records on other Progress platforms.

However, the lock table applies only for Progress 4GL applications. Non-Progress languages (for example, RPG) accessing a Progress/400 dictionary library use the IBM locks LOCK and NO-LOCK. The DataServer issues a DB2/400 lock when it issues a Progress EXCLUSIVE-LOCK. Therefore, the Progress/400 lock table **does not** ensure that records are locked against access by non-Progress applications.

See the "Additional Locking Considerations" section for information about factors that affect record locks in DB2/400. See the chapter on transactions in the *Progress Programming Handbook* for more information on Progress locks.

Creating and Maintaining the Progress/400 Lock Table

The Progress/400 lock table is an OS/400 object of the *USRSPC type with the LOCK attribute. Its name is PROLKT.

The lock table has an initial default value of 500 lock table entries. A lock table entry is required for each currently locked record and for each user (transaction) that is waiting for a locked record (both EXCLUSIVE-LOCK and SHARE-LOCK).

In a Progress database, the lock table exists only during an active Progress session; the Progress/400 lock table object exists until you explicitly delete it, whether or not there is an active Progress session. You are restricted to one lock table per Progress/400 server schema.

To create or modify a lock table, use the Progress/400 Create Lock Table (CRTPROLKT) utility. Use this OS/400 syntax to create a lock table or to change the number of lock table entries:

```
CRTPROLKT PRODCT(library-name) NUMLCK(number)
```

Table 2–6 describes the CRTPROLKT parameters.

Table 2–6: CRTPROLKT Parameters

Parameter	Keyword	View
Progress/400 Dictionary Name	PRODCT	Enter the name of the OS/400 Data Dictionary library where you want the lock table to reside.
Number of lock table entries	NUMLCK	Enter the maximum number of entries that the lock table will support. This number includes both the records that are currently locked and records that are waiting for a record that is locked. The default is 500. The maximum value is 99999.

You can also create a lock table when you run the DUPPRODB utility. Enter *YES for the Create Progress Lock Table parameter. To delete the lock table, delete the OS/400 *USRSPC object.

Standard DB2/400 Locks

Locking on the AS/400 is handled by the DB2/400 record management system. The DB2/400 record management system views a record as having a LOCK or a NO-LOCK. If you use standard DB2/400 locks with Progress applications, an EXCLUSIVE-LOCK is implemented as a DB2/400 LOCK, and SHARE-LOCK and NO-LOCK are implemented as DB2/400 NO-LOCK.

SHARE-LOCK Implications

When you implement standard DB2/400 locks instead of standard Progress locks, you might encounter problems with applications that depend on Progress SHARE-LOCKs. For example, if the program PGM-A has a record under EXCLUSIVE-LOCK, and the program PGM-B requests the same record (SHARE-LOCK), the Progress/400 DataServer allows PGM-B access to the record. You will have a problem if PGM-A updates the record and releases the lock, and PGM-B then updates the record based on the old data. To resolve this problem, explicitly specify EXCLUSIVE-LOCK when you read records for update.

Additional Locking Considerations

This section lists additional considerations about OS/400 locking:

- When you implement Progress locks in the OS/400 environment, the Progress locks apply only for other Progress applications. The DataServer implements the EXCLUSIVE-LOCK as an OS/400 LOCK. The standard OS/400 lock rules apply for non-Progress applications. This is an important factor because OS/400 **locks one record per file**, while the DataServer can read and lock more than one record. The following example illustrates how this can affect your applications:

```
DEFINE BUFFER BUF2 FOR ORDER.
DO:
  FIND ORDER WHERE 1 EQ ORDER-NUM EXCLUSIVE-LOCK.
  FIND BUF2 WHERE 2 EQ BUF2.ORDER-NUM EXCLUSIVE-LOCK.
UPDATE BUF2.
.
.
.
END.
```

  OS/400 locks one record per file except when multiple records are locked in one file, commitment control is active, and a transaction is pending.

Table 2–7 shows how locks for records accessed by this procedure are implemented.

Table 2–7: OS/400 Impact on Progress Applications

Progress Procedure	OS/400 Locks		Progress Locks	
	Record 1	Record 2	Record 1	Record 2
Read record 1 EXCLUSIVE-LOCK	Locked	Unlocked	Locked	Unlocked
Read record 2 EXCLUSIVE-LOCK	Unlocked	Locked	Locked	Locked
Update record 2	Unlocked	Unlocked	Locked	Locked

Progress/400 applications view record 1 as locked throughout the entire procedure, but OS/400 sees record 1 as locked only until record 2 is read.

- OS/400 supports five object-level lock states: EXCLUSIVE (*EXCL), EXCLUSIVE ALLOW READ (*EXCLRD), SHARED FOR READ (*SHRRD), SHARED FOR UPDATE (*SHRUPD), and SHARED NO UPDATE (*SHRNUP). Progress/400 opens the OS/400 physical files as SHARED FOR READ. Once a file is opened, it stays open until you disconnect from the DB2/400 database files. Because the files are in a SHARED FOR READ lock state, Progress/400 might generate an error when you try to run OS/400 utilities that require an incompatible object lock on an open file.

- Each physical file (data file) has a WAITRCD parameter that defines the number of seconds a program waits for a record to be updated or deleted. If the record is not available in the specified time, you get an error message. For most OS/400 files, the WAITRCD default is 60 seconds.

  You can modify the WAITRCD parameter for physical files using the `CHGPF CL` command and for logical files using the `CHGLF CL` command. See the *AS/400 CL Reference* for more information.

- When you use multiple defined buffers in a Progress 4GL procedure, the Progress/400 DataServer opens only one data path to the file, allocates only one cursor, and locks only one record. The Progress 4GL allows a record to be locked for each defined buffer; however, Progress/400 allows only one record per open data path.

- If the AS/400 crashes and records are locked through the Progress lock table, you might need to delete and re-create the table.
- Another technique for avoiding SHARE-LOCK conflicts is to read all records initially with NO-LOCK, then reread them with EXCLUSIVE-LOCK just before committing changes. This technique minimizes the penalty inherent in holding an EXCLUSIVE-LOCK for a long period of time.

2.4 Queries

The Progress 4GL allows you to write database-independent queries. However, you might want to create queries that take advantage of certain DB2/400 features or DataServer performance enhancements.

Progress/400 supports index repositioning.

2.4.1 Joined Logical Files and Progress Queries

DB2/400 supports the joining of logical files. Progress/400 supports these joined files as virtual files. The Progress/400 Data Dictionary has read-only access to these virtual files, so you cannot change or otherwise manage them as you change or manage normal files from the Progress/400 Data Dictionary. See the OS/400 documentation for information on creating joined logical files.

Queries against a joined logical file have certain limitations. Progress uses RECIDs to manage records and improve performance on queries, but RECIDs have little meaning on the AS/400. As a result, queries against joined logical files can return unpredictable results, and some query types might not return the correct set of records.

For example, the following queries work correctly against joined logical files:

```
FOR EACH table NO-LOCK:
FOR EACH table BY key-field-name NO-LOCK:
```

Because the following query contains a nonkeyed field name, it returns unpredictable results when used against joined logical files:

```
FOR EACH table BY non-keyed-field-name:
```

Other queries might or might not work properly. Before you include a query against a joined logical file in an application, make sure to determine whether it works in your environment.

2.4.2 Record Prefetch

For queries built with the FOR EACH statement and the NO-LOCK option, the DataServer prefetches blocks of records and sends them to the client as a single network message. It does this to minimize network traffic and thereby improve performance.

2.5 Field Lists

Progress/400 supports field lists. Field lists can improve performance because the entire record is not returned to the client. Progress returns only the requested fields. However, using field lists in a query does not automatically ensure performance improvements. The following list explains what field lists can do and where they are appropriate:

- Fields lists allow you to select a short list of fields to retrieve in a query. Only the fields listed in the FIELDS clause of DEFINE QUERY are retrieved, thus shortening the data length of the record.
- Field lists are enabled only in a NO-LOCK query. If the query specifies SHARE-LOCK (the default) or EXCLUSIVE-LOCK, field lists are turned off and the entire record is retrieved.
- Specifying a field list in conjunction with a NO-LOCK query enables record prefetch. This allows the Progress/400 DataServer to pack the shortened field list records together, up to the size of the Message Buffer Size (-Mm) startup parameter.

These factors influence performance when using field list queries:

- **Message Buffer Size (-Mm) startup parameter** — To control the size of the network message, use the Message Buffer Size (-Mn) startup parameter when you start the Progress client. For details, see the "Using the Message Buffer Size (-Mm) Startup Parameter" section in Chapter 4, "Remote Access to Progress/400 DataServer."
- **Total number of records** — If a file contains only a few records (less than 100), using field lists does not enhance performance. To see the benefit of field lists, the file must contain enough records (5,000-1,000,000) so that a normal query takes 20 to 30 seconds to run.
- **Record length** — Field list improvements are most dramatic when the record length of a file is close to, or greater than, the Message Buffer Size (-Mm) startup parameter. For example, assume that the file "FILEA" has a record length of 4200 bytes and a -Mm parameter setting of 4000. Since the record length is larger than the -Mm setting, each whole record requires two network packets. The first packet contains the first 4000 bytes of the record, the second contains the remaining 200 bytes. The record is reassembled on

the client. Also, if the file has only a 3000-byte record length, you can still send only one record per network packet. In both of these cases, a field list query can be quite helpful. If, for example, a query requires only two fields (totaling 20 bytes), the DataServer can pack 200 field list records into the 4000-byte network packet.

- **Specifying too many fields in the FIELDS phrase** — If your field list query specifies more than approximately half of the fields in the file, the additional overhead to process the field list outweighs any benefit, thus potentially degrading performance. Generally, you should limit the fields in a query (browser) to just a few and reread specific users' selected records by RECID.

Progress/400 provides the same support for both a DB2/400 database and a Progress database when using field list queries. The following example returns the same result for both a Progress database and a DB2/400 database:

```
DEFINE QUERY myquery FOR customer FIELDS (cust-num name) SCROLLING.
```

Include the SCROLLING option to enable record prefetch. You must include the NO-LOCK option when you open queries with field lists, as in the following example:

```
OPEN QUERY myquery NO-LOCK.
```

Similarly, you must include the NO-LOCK option in FOR EACH statements that include field lists, as in the following example:

```
FOR EACH customer FIELDS (cust-num name) NO-LOCK:
```

Alternatively, you can tune the database to have NO-LOCK as the default. This allows you to use field lists without specifying the NO-LOCK query option.

Use field lists to retrieve only those fields that your application requires. (For performance reasons, the Progress/400 DataServer retrieves the first index field even when you do not include it in the field list. In cases when the DataServer can predict that a query requires a refetch, it retrieves the entire record.) The DataServer allocates memory based on the maximum size specified for a field in a record. Omitting larger fields from a query enhances performance.

When the DataServer processes a query with a field list, it caches the fields that are part of the field list and any other fields that the query specified, which you can then access without making another call to the DB2/400 RDBMS. For example, the Progress/400 DataServer fetches the name and the zip field to process the following query:

```
FOR EACH customer FIELDS (name) WHERE zip = 01730 NO-LOCK:
```

See the Record Phrase entry in the *Progress Language Reference* for more information on the FIELDS option.

2.6 Language Restrictions

Because the 4GL is database independent, most 4GL language elements (statements, functions, and so forth) work the same whether your application accesses a DB2/400 database or a Progress database. The following sections describe the Progress 4GL statements or functions that are not supported or that return different results when used with the Progress/400 DataServer against a DB2/400 database. Most of the statements perform Data Dictionary maintenance or transaction management (database recovery) functions. Use OS/400 facilities to perform these tasks. If you use these statements, the errors they generate are not fatal unless specifically noted.

For language restrictions using the native clients, see the "Executing Progress Code from the Native Clients" section in Chapter 5, "Preparing to Use AS/400-based Clients."

2.6.1 ALTER TABLE, CREATE INDEX, CREATE TABLE, CREATE VIEW, DROP INDEX, DROP TABLE, DROP VIEW Statements

These statements cause compile-time and run-time errors when you use them with a DB2/400 database. Use OS/400 facilities or the Progress/400 Data Dictionary tool for schema maintenance.

2.6.2 DBNAME Function

The DBNAME function returns the name of the first connected database, usually the client schema-holder name.

2.6.3 GRANT Statement

The GRANT statement is not supported and its use causes a compile-time error. You must use the appropriate `CL` command to manipulate OS/400 authorities and specify permissions.

2.6.4 ROWID Function

In Progress, you can create a ROWID without creating a record. In DataServer applications, creating a ROWID creates a record. The following statement illustrates the difference in behavior:

```
CREATE customer.
a=ROWID (customer).
```

The DataServer creates a customer record using default values. After the user assigns values to the fields in that record, the DataServer updates it. When you UNDO the transaction, the DataServer deletes the record.

If you have a unique index on that file, two users cannot access the default value of the indexed field simultaneously.

The value returned by the ROWID function is the DB2/400 relative record number (RRN) of the database record currently associated with the specified record buffer. Therefore, the value is not unique across all records in the database, but it is unique across all records of its associated file. You can use ROWID in Progress/400 as you do in standard Progress with the following additional restrictions:

- In standard Progress, when you delete a record and then UNDO the delete transaction, the ROWID value remains the same because Progress reserves the ROWID value in the database until it commits the transaction.
- In Progress/400, the ROWID value can change during a delete transaction because the record is deleted from the database before the transaction is committed, and without holding a place in the database. When the rollback (UNDO) occurs, the record is added back into the file.
- For virtual table ROWID considerations, see the "Virtual Tables" section.
- ROWID provides the same functionality as RECID, but ROWID is more consistent across data sources.

2.6.5 RECID Function

The information in the "ROWID Function" section applies to the RECID function supported by the current Progress clients.

2.6.6 REVOKE Statement

The REVOKE statement is not supported and its use causes a compile-time error. You must use the appropriate CL command to manipulate OS/400 authorities and specify permissions.

2.6.7 SETUSERID Function

The SETUSERID function always returns false with the Native 4GL Client because you cannot change a user profile on the AS/400 with the Progress 4GL. With other clients, the SETUSERID function applies only to the user ID and password for the client schema holder. Use the User ID (-U) and Password (-P) connection parameters when you are connecting to DB2/400 database files, whether or not they exist in the client schema holder's _User file. SETUSERID returns false if there is no _User file. Using SETUSERID for the schema holder does not change the user ID for the DB2/400 database files you connected to.

See also the "USERID Function" section.

2.6.8 USERID Function

The USERID function returns the OS/400 user profile of a job when used with the Native 4GL Client. With other clients, USERID returns the User ID (-U) parameter specified during the connect even if there is no _User file for the schema holder.

See also the "SETUSERID Function" section.

2.7 Application Security

In standard Progress, the Data Dictionary handles both database and application security. When you use Progress/400 with DB2/400 database files, you use OS/400 security facilities. This section briefly describes security features that you might want to consider in addition to the OS/400 features. SeeChapter 8, "System Administration," for more information on how the DataServer implements security in the OS/400 environment. Also, see the *AS/400 Security Concepts and Planning Guide*.

Code your applications to check for a user ID. Once a user enters identification, your procedures can verify that the user is authorized to run individual procedures within the application. However, if your application runs on many different machines with many users, you probably do not want to specify user IDs in your procedures. In this case, build a database file on the AS/400 that contains security information and can be accessed before running specific applications.

2.7.1 Setting Up a User Permission File on the AS/400

Define a database file that contains a record for each of the different procedures within your application and lists the users authorized to run each activity. The application can read the database file to verify that the user ID established at startup is in the list of authorized users for that activity.

Because this information is stored in the database and not specified in your application, the procedure remains the same when the permission information changes.

2.7.2 Defining a User ID and Password at Startup

For security purposes, you might prefer to prompt the user for the user ID and password rather than use the User ID (-U) and Password (-P) connection parameters. The following code is an example of a routine that uses a user-supplied login to connect to DB2/400 database files:

```
DEFINE VARIABLE id AS CHAR FORMAT "x(8)" INITIAL [" "].
DEFINE VARIABLE password AS CHAR FORMAT "x(8)" INITIAL [" "].
DEFINE VARIABLE hostid AS CHAR FORMAT "x(10)" INITIAL [" "].
DEFINE VARIABLE args AS CHAR FORMAT "x(40)".

UPDATE SPACE (2) hostid SKIP SPACE (2) id LABEL "UserID" SKIP password
BLANK WITH CENTERED ROW 8 SIDE-LABELS ATTR-SPACE
TITLE " Login to Database  ".

args = " -H " + hostid + " -U " + usrid + " -P " + password + -N TCP.

HIDE ALL.

/* Connect to the database schema holder */

CONNECT  schmhldr -1.

/* Connect to the DB2/400 DataServer database */

CONNECT as400db VALUE(args).
```

2.8 Progress/400 Word Index Support

Progress/400 supports word indexing. This feature is similar to standard Progress RDBMS word indexing, but is implemented using features and functions that are unique to the AS/400. For more information on using Progress word indexes, see the *Progress Programming Handbook*.

The following restrictions apply to the use of Progress/400 word indexing:

- You cannot build a Progress/400 word index over a multi-member physical file.
- You cannot build a Progress/400 word index over DDM files.
- If you build a Progress/400 word index over a physical file that uses DB2/400 triggers, the DB2/400 triggers are replaced with Progress/400 triggers.

The following section explains how to set up and use Progress/400 word indexing. Subsequent sections provide a technical discussion of word indexing and discuss coding considerations.

2.8.1 Setting Up and Using Word Indexes

This section provides quick-start instructions for the following tasks:

- Setting up a word index on an existing or new Progress/400 Dictionary Library
- Handling a renamed Progress/400 Dictionary Library
- Changing word-break rules for an existing Progress/400 Dictionary Library

These tasks use a number of server utilities. For details, see Chapter 10, "AS/400 Utilities."

Once you have built a word index, you use the Progress/400 Data Dictionary to maintain it. For details, see Chapter 9, "Remote Client DB2/400 Utilities."

Setting Up Word Indexes on an Existing Progress/400 Dictionary Library

Follow these steps to set up a word index on an existing Progress/400 Dictionary Library:

1 ♦ If the default word-break table (named DFTWISWST) is not adequate for your needs:

   a) Create a new word-break table using the UPDPROWBT utility.

   b) After the table has been created, replace the Dictionary's word-break table using the RPLDCTWBT utility.

2 ♦ Use the Progress/400 Data Dictionary to create one or more word indexes over one or more tables.

   NOTE: If you create word indexes over journaled tables, you must start the Progress/400 Word Index Support Processor using the STRWISPRC utility before you can perform any file update operations.

You can now use your word indexes.

Setting Up Word Indexes on a New Progress/400 Dictionary Library

Follow these steps to set up a word index on a new Progress/400 Dictionary Library:

1 ♦ If the default word-break table (named DFTWISWST) is not adequate for your needs, create a new word-break table using the UPDPROWBT utility.

2 ♦ Use the DUPPRODB to create a new Progress/400 Dictionary Library. If you created a new word-break table in Step 1, be sure to specify it on the DUPPRODB command line.

3 ♦ Use the Progress/400 Data Dictionary to either add tables, indexes, and so forth, or load your definition (.df) file.

Loading a definition file that contains word indexes creates an index with a default word size of 30. You can change this size prior to committing by using the Index Property screen in the Progress/400 Data Dictionary. See the "Modifying an Index" section in Chapter 9, "Remote Client DB2/400 Utilities," for details.

NOTE: If you create word indexes over tables that will be journaled, you must start the Progress/400 Word Index Support Processor using the STRWISPRC utility before you can perform any file update operations.

You can now use your word indexes.

Handling a Renamed Progress/400 Product Library

If your Progress/400 Product Library is renamed after installation and creation of your word indexes, follow these steps:

1 ♦ Add the renamed Progress/400 Product Library to your library list.

2 ♦ Use the `UPDWISTRG` command to update the library of the trigger program for each physical file that has word indexes.

Changing the Word-Break Rules for an Existing Progress/400 Dictionary Library

Follow these steps to change the word-break rules for an existing Progress/400 Dictionary Library:

1 ♦ Use the UPDPROWBT utility to create a new word-break table.

2 ♦ After the table has been created, use the RPLDCTWBT utility to update the word index definitions in the Progress/400 Dictionary Library.

Running this utility rebuilds each word index using the new table's rules from that Dictionary Library.

2.8.2 How Progress/400 Maintains Word Indexes

Progress/400 word indexes are maintained by a Progress-supplied trigger program and the Progress-supplied Word Index Support Processor. When you create a word index for a physical file, triggers are added to the file using the OS/400 ADDPFTRG (Add Physical File Trigger) command. The Progress PROWISTRG program then becomes the trigger program for after-insert, after-delete, and after-update events. This ensures the maintenance of any Progress/400 word indexes defined for a physical file whenever the file is changed. DB2/400 triggers are used because they greatly minimize the impact on your existing AS/400 applications. DB2/400 triggers provide a method that Progress uses to "hook" into your file updates.

DB2/400 triggers are fired before or after a record is created, updated, or deleted. This allows the Progress/400 trigger program to be called whenever a word index update is required. Triggers are not fired when a transaction is rolled back using any OS/400 ROLLBACK operation or the Progress UNDO statement; however, Progress handles all ROLLBACKs properly. The process used to maintain Progress/400 word indexes depends on whether commitment control is in effect:

- If commitment control **is not** in effect, the word index update is performed in the trigger program before control is returned to the application program. This operation is synchronous. Therefore, control is not returned to your program until the trigger program completes its work.

- If commitment control **is** in effect, the trigger program updates the Progress-supplied Trigger Transaction file and the Word Index Support Processor, then performs the actual word index updates. Control is returned to your program after the Trigger Transaction file record is written.

NOTE: If the trigger program detects that commitment control is active but the Word Index Support Processor is not running, the trigger program sends an escape message to the application program indicating that the trigger program failed. If your application does not handle this escape message properly, your application might fail. **If your application uses commitment control, files that are opened under commitment control cannot be updated if the Word Index Support Processor is not running**.

The Word Index Support Processor

The Progress/400 Word Index Support Processor updates Progress/400 word indexes if the physical file is opened for update with commitment control active. The Word Index Support Processor consists of two perpetual jobs that are started using the Progress-supplied STRWISPRC utility. (For details, see its description in Chapter 10, "AS/400 Utilities.") Once these jobs are running, they process all transaction COMMITs that your application performs that affect DB2/400 files with Progress/400 word indexes built over them:

- The PROWISRCV job handles the initial processing of transaction COMMITs.
- The PROWISDTAQ job handles the actual update of Progress/400 word indexes.

These jobs use a Progress/400 library called the Word Index Work Library.

NOTE: If the AS/400 crashes, you might not be able to restart the word index processor. If this occurs, call Progress Technical Support.

The Word Index Work Library

The Word Index Work Library is created when Progress/400 is installed on your AS/400. You are prompted for a library name (the default is PROWISWRK), then the install process creates the named library and places into it the objects required by the Word Index Support Processor.

NOTE: **Do not** rename, delete, move, or change these objects in any way or word indexing will not work.

The objects in the Word Index Work Library include the following:

- The Progress/400 Trigger Transaction file
- The PROWISJRN journal and its receivers
- The PROWISCFG user space
- The PROWISDTAQ data queue

These objects are static and are always in the Work Library. In addition, some dynamic objects are placed in this library while the Word Index Support Processor is running.

Multiple installations of Progress/400 can share the Word Index Work Library.

Preventing Word Index Corruption

Certain OS/400 commands will corrupt a Progress/400 word index or cause Progress/400 word indexing not to function when they are run against:

- A physical file with Progress/400 word indexes built over it
- The Progress/400 install library

If you run these commands, you must follow specific recovery procedures to fix the problems that they cause. Table 2–8 documents the commands, the resulting problems, and the recovery procedures.

Table 2–8: Word Index Corruption Recovery Procedures *(1 of 2)*

OS/400 Command	Problem	Recovery Procedure
RGZPFM	Can change the relative record number of a record.	Run the Progress/400 GENWRDIDX utility to generate the word indexes of any affected file.
APYJRNCHG	Can change a file's records without firing the file's triggers.	Run the Progress/400 GENWRDIDX utility to generate the word indexes of any affected file.
RMVJRNCHG	Can change a file's records without firing the file's triggers.	Run the Progress/400 GENWRDIDX utility to generate the word indexes of any affected file.
RMVPFTRG	Removes the trigger program required by Progress/400.	Using the Progress/400 Data Dictionary, delete the word indexes of the affected file, COMMIT, then re-create the word indexes.
ADDPFTRG	Changes the trigger program required by Progress/400.	Using the Progress/400 Data Dictionary, delete the word indexes of the affected file, COMMIT, then re-create the word indexes.

Table 2–8: Word Index Corruption Recovery Procedures *(2 of 2)*

OS/400 Command	Problem	Recovery Procedure
RSTOBJ or RSTLIB to restore a physical file.	Can change a file's records without firing the file's triggers.	Run the Progress/400 GENWRDIDX utility to generate the word indexes of any affected file.
RSTLIB to restore the Progress/400 Product Library into a different library.	Because the original library no longer exists, the trigger program is not found when a file with word indexes built over it is opened.	Add the new Progress/400 Product Library to your library list, then run the UPDWISTRG utility to change trigger programs for any files in that library that have word indexes built over them.

For descriptions of the utilities noted in the recovery procedures, see Chapter 10, "AS/400 Utilities."

2.8.3 Coding Issues

This section documents coding issues that you must consider when developing an application that uses Progress/400 word indexes.

Most issues involve minor Progress 4GL syntax differences. You generally code Progress 4GL applications that use Progress/400 word indexes identically to those that use standard Progress word indexes; however, there are a few cases when you must use slightly different syntax to achieve the desired result.

WHERE Clause AND Support

The following Progress 4GL program will successfully retrieve records from a table in a Progress database that uses word indexes:

```
FOR EACH customer WHERE comments CONTAINS "credit"
AND credit CONTAINS "hold":
  DISPLAY cust-num comments.
END.
```

Note that this program uses the AND keyword between the CONTAINS clauses. Remote Progress clients, Version 9.0A or earlier, cannot handle a query of this type. However, you can

work around this problem by substituting the following code, which omits the AND keyword and uses normal CONTAINS syntax:

```
FOR EACH customer WHERE comments CONTAINS "credit & hold":
  DISPLAY cust-num comments.
END.
```

Extent Fields

The Progress/400 DataServer supports extent fields just as the Progress database does; however, DB2/400 does not support extents or arrays in DB2/400 physical files. Extent support is provided by placing each element of an extent field contiguously in the physical file's record format. For example, suppose that you use the Progress/400 Data Dictionary to create a table named TBL1 that contains the following fields:

Field Name	Data Type	Extents
Cust-Num	Integer	0
Name	Character	0
Comment	Character	6

The DDS generated on the AS/400 for this table follows:

Field Name	Data Type	Length
CUST-NUM	Binary	4
NAME	Character	20
Comment01	Character	25
Comment02	Character	25
Comment03	Character	25
Comment04	Character	25
Comment05	Character	25
Comment06	Character	25

Observe that the single six-extent Comment field of TBL1 has been converted to a set of six Comment*xx* fields in the DDS. These Comment*xx* fields are unknown to the Progress client. Each Comment*xx* field represents one element of the Progress extent field, and the fields exist in the record format contiguously. When you build a word index using the Comment field, all of the Comment*xx* fields are considered to be a single large character file or buffer, and words are pulled from the entire buffer to generate the index.

This conversion of extent fields for AS/400 might cause some unexpected results when querying the file. Since the entire buffer consisting of all of the fields is considered to be the extent field, word separation might be an issue. When the Comment field is updated using the Progress UPDATE statement, each extent of the field is a distinct field; however, when the data entered is sent to the AS/400, it is stored in a single large buffer, and the individual elements of the EXTENT field are not known. This is not true in a Progress database, where each element of the EXTENT field is considered to be a separate field.

Suppose, for example, that:

- The fields Comment[1] and Comment01 contain the following string:

  TEST OF EXTENT WORD BREAK

- The fields Comment[2] and Comment02 contain the following string:

  THIS IS A WORD BREAK TEST

Now, suppose that you execute the following query:

```
FOR EACH tbl1 WHERE comment CONTAINS "THIS":
  DISPLAY comment.
END.
```

The query does not find THIS because the text in the Comment buffer is as follows:

TEST OF EXTENT WORD BREAKTHIS IS A WORD BREAK TEST

Record Locking

Normal record-locking rules apply when performing query-by-word queries.

3

Creating the Progress/400 Environment

This chapter describes how to create the components that allow you to access databases. Specifically, it discusses:

- Creating the Progress/400 environment on the AS/400
- Changing DB2/400 data definitions to Progress equivalents
- Creating and deploying a client schema holder
- Maintaining DB2/400 data definitions
- Programming considerations
- Special DB2/400 considerations

3.1 Overview

Progress/400 allows you to use Progress applications to access data stored in a DB2/400 database on the AS/400. To do this, you must create a Progress environment on the AS/400. The Progress environment is where the data definitions for the DB2/400 database files will be stored in a format that Progress can read. These data definitions will be contained in a series of files whose names begin with P__. These files reside in the server schema contained in a single library, although your DB2/400 database files might be distributed across many libraries.

Other components that make up the Progress/400 environment are the client and its schema holder. Accessing DB2/400 database files through the DataServer involves migrating the data definitions into the Progress/400 environment on the AS/400 and creating a schema holder on the client with those same data definitions so that they can be read by Progress applications. The client component also includes the Progress/400 Data Dictionary tool that allows you to maintain DB2/400 data definitions from the Progress client. All of your development can be done in a single environment. However, you still have the option of maintaining the DB2/400 data definitions by means of DDS on the AS/400. Figure 3–1 shows the client and server components of the Progress/400 environment.

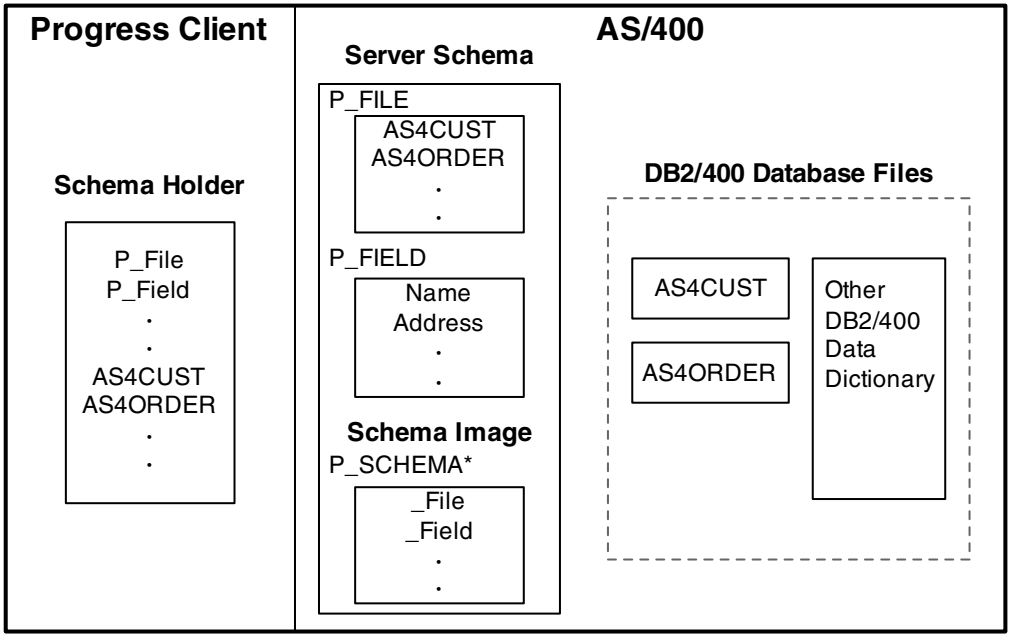


Figure 3–1: The Progress/400 Environment

3.2 Accessing an Existing DB2/400 Database

These are the tasks involved in creating the Progress/400 environment on the AS/400 and preparing your DB2/400 objects for access by the Progress client through the DataServer:

- Establish basic connectivity between the client PC and the AS/400.
- Back up your DB2/400 database files.
- If the DB2/400 database files use a code page other than IBM037, make sure that you have the appropriate ALTSEQ tables to handle case issues. See the "User-defined Collation Tables" section in Chapter 8, "System Administration," for a discussion of issues to consider.
- Run DUPPRODB to create the empty server schema on the AS/400 machine.
- Run CHGPRODCT to populate the schema files with data definitions for the DB2/400 database files.
- Create and synchronize a client schema holder.
- Adjust your code to account for differences in behavior between the Progress RDBMS and DB2/400.
- Decide how you will maintain the DB2/400 data definitions: either through DDS on the AS/400 machine or through the Progress/400 Data Dictionary on the client PC. Do not mix the two methods.
- If you use DDS to maintain data definitions, update the server schema with CHGPRODCT to reflect any modifications that you make to the DB2/400 definitions.

3.2.1 Creating the Environment (DUPPRODB)

Running the DUPPRODB utility on the AS/400 is the first step in creating the Progress/400 environment. The DUPPRODB utility creates a library that contains the server schema and the DB2/400 database files that you select to include.

Follow these steps to set up the Progress/400 environment on the AS/400:

1 ♦ Make sure that you have basic connectivity between the AS/400 and your PC client.

2 ♦ Back up your DB2/400 database files.

3 ♦ Type **DUPPRODB** at the OS/400 command line, then press **F4** to display the utility's parameters.

4 ♦ Press **F10** to view more parameters. For a complete description of all of the parameters, see the "Duplicate Progress/400 Database (DUPPRODB)" section in Chapter 10, "AS/400 Utilities."

The DUPPRODB utility creates the Progress/400 environment on the AS/400 machine and builds the structure for the server schema. After you run the DUPPRODB utility, run the CHGPRODCT utility as described in the "Changing Data Definitions (CHGPRODCT)" section to re-create the data definitions for your DB2/400 database files. The utility stores those data definitions in the server schema in the Progress/400 environment that you created with the DUPPRODB utility.

3.2.2 Changing Data Definitions (CHGPRODCT)

Running the CHGPRODCT utility accesses the physical and logical files that make up the set of your DB2/400 database files. It re-creates the necessary data definitions in the server schema in the Progress/400 environment so that the Progress client can access them through the DataServer. For example, it converts the DDS zoned numeric data type to the Progress DECIMAL data type. The utility can process data definitions that were made by DDS or SQL.

The physical and logical files can reside in multiple libraries, but CHGPRODCT places the schema for these files in a single library, the library that you specified with DUPPRODB.

Follow these steps to run CHGPRODCT:

1 ♦ Type **CHGPRODCT** at the OS/400 command line, then press **F4** to display the utility's parameters.

2 ♦ Complete the parameters. For a complete description of all of the parameters, see the "Change Progress/400 Dictionary Utility (CHGPRODCT)" section in Chapter 10, "AS/400 Utilities."

The CHGPRODCT utility translates DB2/400 definitions into Progress definitions on the AS/400 and stores these in files with a P__ prefix. The Progress/400 server schema now contains the data definitions for the DB2/400 database files that you want to access through a DataServer application.

3 ♦ Since the next task requires accessing the AS/400 from the client machine, make sure that you set up the appropriate network connections and start any supporting processes on the AS/400. Table 3–1 briefly lists what you must do for each protocol. See Chapter 4, "Remote Access to Progress/400 DataServer," for more details.

Table 3–1: AS/400 Network Processes

Protocol	Startup
SNA	There are no processes that you start manually.
TCP/IP	Start the TCP/IP broker with the `STRPRONET` command.

Next, you must set up the client component of the Progress/400 environment.

3.2.3 The Client Schema Holder

The server schema on the AS/400 and the client schema holder are key components in the Progress/400 architecture. These components allow Progress 4GL applications to access DB2/400 database files by storing data definitions in an accessible format. WebSpeed and Open AppServer agents are clients to the Progress/400 DataServer, hence the information in this section also applies to them.

Creating a client schema holder involves the following general steps:

- Create an empty Progress database on the client machine.
- Run the Create DataServer Schema utility on the client, which copies the data definitions of the DB2/400 database files.

The schema holder need not reside on the client machine. For example, for ease of maintenance, you can locate the schema holder on a file server that multiple clients can access.

NOTE: A single schema holder can hold schemas for several non-Progress databases. In addition, it can hold the schema for one Progress database.

Creating an Empty Progress Database

The Progress/400 environment uses the empty database on the client to create a schema holder for your DB2/400 data definitions.

There are four ways to create an empty Progress database:

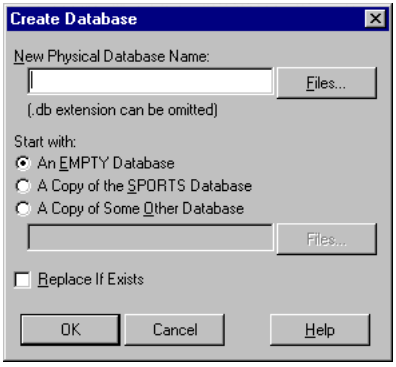
- With the PRODB utility
- With the CREATE DATABASE statement
- With the Data Dictionary
- With the Data Administration tool

See the *Progress Database Administration Guide and Reference* for information on these techniques.

This section describes how to create a database with the Data Administration tool. Follow these steps to create and connect an empty Progress database on your client machine:

1 ♦ Start the Progress desktop with no database connected and access the Data Administration tool.

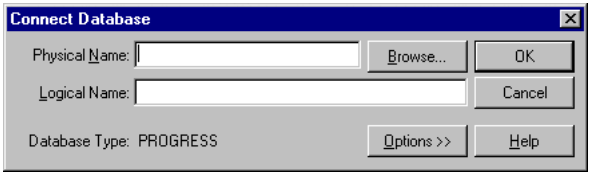
2 ♦ From the main menu, choose Database→ Create. The Create Database dialog box appears:



3 ♦ Type the schema-holder name (for example, as4sh) in the New Physical Database Name field. This name must be different from the server schema library on the AS/400.

4 ♦ Activate the An EMPTY Database radio button.

5 ♦ Choose OK. The Connect Database dialog box appears. By default, the name of the newly created database appears in the Physical Name field.



You do not need to provide any additional connection information. You can add connection parameters when you create the database or edit connection information later. See the online help for a complete description of the Connect Database dialog box.

If you choose to add a logical database name in the Logical Name field, use the logical database name to refer to the database in your programming applications. For more information on database names, see the database access chapter in the *Progress Programming Handbook*.

6 ♦ Choose OK to connect the empty Progress database and return to the Data Administration main window.

Next, run the Create DataServer Schema utility.

Creating the Client Schema Holder

Before you can create the client schema holder, you must:

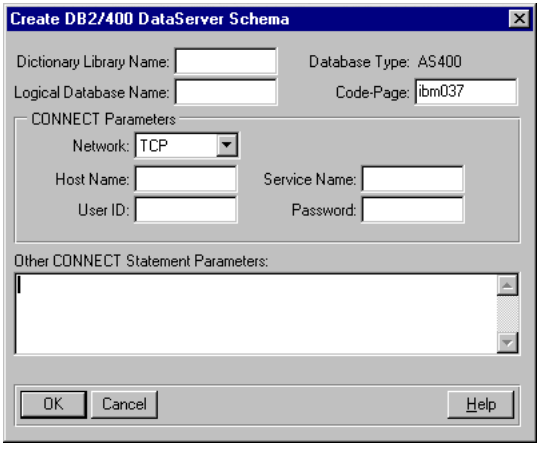
- Create the Progress/400 server schema on the AS/400.
- Add the data definitions for your DB2/400 database files to the server schema.
- Start the processes required by your network protocol.
- Create an empty Progress database on the client machine.

Follow these steps to create the schema holder:

1 ♦ Connect to an empty Progress database.

2 ♦ From the Data Administration main menu, select DataServer→ DB2/400 Utilities→ Create DB2/400 DataServer Schema.

The following dialog box appears:



3 ♦ Type the name of the library that contains the server schema in the Dictionary Library Name field.

4 ♦ Add logical database name information. A logical database name is a database reference that represents the name of a connected physical database. Progress uses the logical database name to resolve database references. When a procedure is compiled against a database, Progress stores the logical database name in the procedure's r-code. When a procedure executes, its database references must match the logical name of a connected database.

5 ♦ In the Code-Page field, type the name of the code page that the DB2/400 database files use (for example, IBM500). By default, the code page is IBM037.

If your DB2/400 database files use a code page that Progress does not support, you must supply a conversion table in the `CONVMAP.DAT` file on the client. This file translates between the Progress client code page and the code page that the DB2/400 database files use.

See the "User-defined Collation Tables" section in Chapter 8, "System Administration," for more information on how the DataServer handles code pages and code-page conversion tables. For a complete discussion of code pages, see the *Progress Internationalization Guide* and the *AS/400: National Language Support Planning Guide*.

6 ♦ Enter the information that your network protocol requires, as Table 3–2 describes.

Table 3–2: Network Connection Parameters

Protocol	Required Connection Parameters
SNA	Host Name: *host-name* Service Name: as400sna Service Name: *server-name.progress-library-name* Username: *username* Password: *password*
TCP/IP	Host Name: *host-name* Network: tcp Service Name: *service-name* Username: *username* Password: *password*

See the "Connecting at Startup" section in Chapter 4, "Remote Access to Progress/400 DataServer," for a description of the required and optional connection parameters.

This connection information is stored with the schema holder. Progress uses the parameters that you provide here whenever you connect to the schema holder though the Data Dictionary or Data Administration tool, or reference a file within the DB2/400 database.

7 ♦ Choose OK. After the utility loads the data definitions of the P__ files, you can choose to synchronize the client schema holder with the server schema.

8 ♦ Choose OK. A message appears when the synchronization is complete.

9 ♦ Choose OK.

The Progress Data Administration tool provides a set of DB2/400 utilities that you can use to maintain a client schema holder. See Chapter 9, "Remote Client DB2/400 Utilities," for descriptions of these utilities.

Deploying a Schema Holder

The guidelines and techniques that apply to deploying a Progress database also apply to deploying a schema holder for DB2/400 database files. However, you must make changes to **both** the supporting DB2/400 database files and the schema holder. There are two techniques for updating a schema holder:

- Allow your users to use the Synchronize Progress/400 Client utility.
- Supply a new schema holder with the P__files and incorporate the `as4sync.p` procedure into your application. See Chapter 9, "Remote Client DB2/400 Utilities," for information on this procedure.

3.2.4 Maintaining DB2/400 Data Definitions

The Progress/400 environment is not a static one. It supports two techniques for modifying the DB2/400 data definitions:

- From the client using the Progress/400 Data Dictionary. Use this technique only from Progress clients running in MS-Windows. UNIX clients do not support the Progress/400 Data Dictionary. See Chapter 9, "Remote Client DB2/400 Utilities," for information on using the Progress/400 Data Dictionary.

  CAUTION: Do not use the Progress/400 Data Dictionary if your DB2/400 database files contain DDS definition attributes that are not supported by the Progress/400 DataServer. Any attributes that are not supported will be lost.

- On the AS/400 using DDS or SQL. Use this technique if your DB2/400 database files contain data that both Progress and AS/400 high-level languages (HLL) applications access.

When you make changes with DDS or SQL, do the following to make sure that the changes are reflected throughout the Progress/400 environment:

1. Run the CHGPRODCT utility to update the Progress/400 environment with those changes. You must run this utility even if the only change you make is to move a DB2/400 file to another library.

2. On the client machine, run the Synchronize Progress/400 Client utility to synchronize the client schema holder so that it reflects the changes that you made on the AS/400, or incorporate the `as4sync.p` procedure into your application. You must do this for each client.

3.2.5 Programming Considerations

Before you can access the objects in the Progress/400 environment with a Progress application, you should account for differences in behavior between working with a Progress database and working with DB2/400 database files. For topics to consider see:

- Database objects
- Data types
- ASCII and EBCDIC sort orders
- Locking behavior
- Unsupported Progress statements and functions

See Chapter 2, "Common Product Information," for a discussion of these topics.

3.3 Moving a Progress Database to the AS/400

These are the tasks involved in moving a Progress application to the AS/400:

- Establish basic connectivity between the client machine and the AS/400.
- Run DUPPRODB to create the empty server schema on the AS/400 machine.
- Dump data and data definitions from your Progress database.

- Create and synchronize a client schema holder.
- Load the data definitions from your Progress database into the Progress/400 environment.
- Synchronize the client schema holder.
- Load data into data files on the AS/400.
- Adjust your code to account for differences in behavior between Progress and DB2/400.
- Decide how you will maintain the DB2/400 data definitions: either through DDS on the AS/400 machine or through the Progress/400 Data Dictionary on the client machine. Do not mix the two methods.
- If you use DDS to maintain data definitions, update the server schema with CHGPRODCT to reflect any modifications that you make to the DB2/400 definitions.

3.3.1 Creating the Environment (DUPPRODB)

Running the DUPPRODB utility on the AS/400 is the first step in creating the Progress/400 environment. The DUPPRODB utility creates an empty server schema that will hold your Progress data definitions.

Follow these steps to set up the Progress/400 environment on the AS/400:

1 ♦ Make sure that you have basic connectivity between the AS/400 and your client.

2 ♦ Type **DUPPRODB** at the OS/400 command line, then press **F4** to display the utility's parameters.

3 ♦ Press **F10** to view more parameters. Complete the parameters. For a complete description of all of the parameters, see the "Duplicate Progress/400 Database (DUPPRODB)" section in Chapter 10, "AS/400 Utilities."

The DUPPRODB utility creates the Progress/400 environment on the AS/400 machine and builds the structure for the server schema.

4 ♦ Since subsequent tasks require accessing the AS/400 from the client machine, make sure that you have set up the appropriate network connections and started any supporting processes. Table 3–3 briefly lists what you need to do for each protocol. See Chapter 4, "Remote Access to Progress/400 DataServer," for more details.

Table 3–3: AS/400 Network Processes

Protocol	Startup
SNA	There are no processes that you start manually.
TCP/IP	Start the TCP/IP broker with the `STRPRONET` command.

Next, you will run client utilities to dump and load Progress data definitions and data into the Progress/400 environment.

3.3.2 Dumping Data Definitions and Data

After you create a Progress/400 environment on the AS/400, you can move your Progress data definitions to it. The first step is to dump the data and data definitions from your Progress database.

Follow these steps on your client machine:

1 ♦ Start Progress, connect to your database, and access the Data Administration tool.

2 ♦ From the Data Administration main menu, choose Admin→ Dump Data and Definitions→ Data Definitions (.df file) to dump the data definitions from your database or from specific tables in your database.

3 ♦ If you are also moving data to the AS/400, choose Admin→ Dump Data and Definitions→ Table Contents (.d file) to dump the data from your database or from specific tables in your database.

Next, you must create a client schema holder so that you can access your data definitions on the AS/400 with Progress client applications through the DataServer.

3.3.3 The Client Schema Holder

The server schema on the AS/400 and the client schema holder are key components in the DataServer architecture. These components allow client Progress applications to access objects on the AS/400.

Creating a client schema holder involves the following general steps:

- Create an empty Progress database on the client machine.
- Run the Create DataServer Schema utility on the client, which loads the data definitions of the P__ files into the client schema holder.

The schema holder need not reside on the client machine. For example, for ease of maintenance, you can locate the schema holder on a file server that multiple clients can access.

Creating an Empty Progress Database

The Progress/400 environment uses the empty database on the client to create a schema holder for your DB2/400 data definitions.

An empty database on the client machine serves as a schema holder that the Progress/400 DataServer uses to access the data definitions that you moved to the AS/400.

There are four ways to create an empty Progress database:

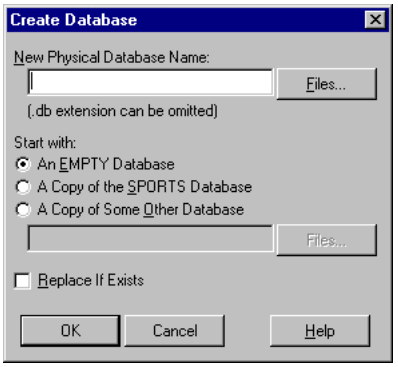
- With the PRODB utility
- With the CREATE DATABASE statement
- With the Data Dictionary
- With the Data Administration tool

See the *Progress Database Administration Guide and Reference* for information on these techniques.

This section describes how to create an empty database with the Data Administration tool. Follow these steps to create and connect an empty Progress database on your client machine:

1 ♦ Start Progress with no database connected and access the Data Administration tool.

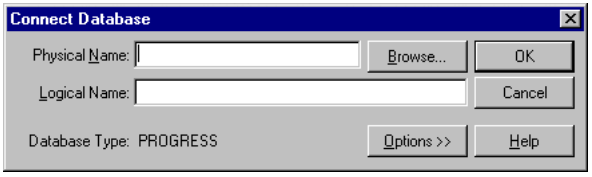
2 ♦ From the main menu, choose Database→ Create. The Create Database dialog box appears:



3 ♦ Type the schema-holder name (for example, as4sh) in the New Physical Database Name field.

4 ♦ Activate the An EMPTY Database radio button.

5 ♦ Choose OK. The Connect Database dialog box appears. By default, the name of the newly created database appears in the Physical Name field:



You do not need to provide any additional connection information. You can add connection parameters when you create the database or edit connection information later. See the online help for a complete description of the Connect Database dialog box.

If you choose to add a logical database name in the Logical Name field, use the logical database name to refer to the database in your programming applications. For more information on database names, see the database access chapter in the *Progress Programming Handbook*.

6 ♦ Choose OK to connect the empty Progress database and return to the Data Administration main window.

Next, run the Create DataServer Schema utility, as the "Creating the Client Schema Holder" section describes.

Creating the Client Schema Holder

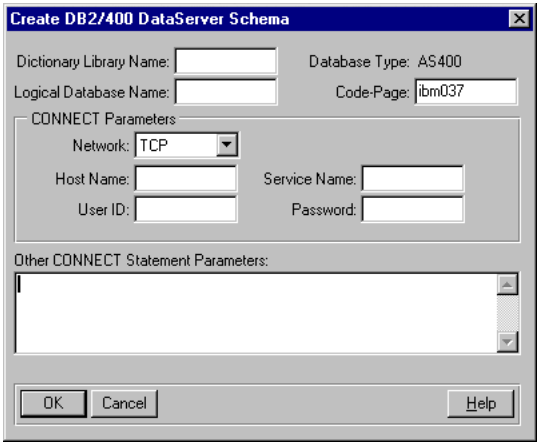
Before you can create the client schema holder, you must:

- Create an empty Progress/400 server schema on the AS/400.
- Start the processes required by your network protocol.
- Create an empty Progress database on the client machine.

Follow these steps to create the schema holder:

1 ♦ Connect to an empty Progress database.

2 ♦ From the Data Administration main menu, select DataServer→DB2/400 Utilities→Create DB2/400 DataServer Schema. The following dialog box appears:



3 ♦ Type the name of the library that contains the empty server schema in the Dictionary Library Name field.

4 ♦ Enter the logical database name.

5 ♦ In the Code-Page field, type the name of the code page that the DB2/400 database files that contain your Progress definitions use. By default, the code page is IBM037.

If your DB2/400 database files will use a code page that Progress does not support, you must supply conversion tables that translate between the Progress client code page and the

code page that the DB2/400 database files use. Entries for these tables must be placed in the `CONVMAP.DAT` file for the client.

See the "User-defined Collation Tables" section in Chapter 8, "System Administration," for more information on how the DataServer handles code pages and code-page conversion tables. For a complete discussion of code pages, see the *Progress Internationalization Guide* and the *Application System/400 National Language Support Planning Guide*.

6 ♦ Enter the information that your network protocol requires, as Table 3–4 describes.

Table 3–4: Network Connection Parameters

Protocol	Required Connection Parameters
SNA	Host Name: *host-name* Service Name: as400sna Service Name: *server-name.progress-library-name* Username: *username* Password: *password*
TCP/IP	Host Name: *host-name* Network: tcp Service Name: *service-name* Username: *username* Password: *password*

See the "Connecting at Startup" section in Chapter 4, "Remote Access to Progress/400 DataServer," for a description of the required and optional connection parameters.

This connection information is stored with the schema holder. Progress uses the parameters that you provide here whenever you connect to the schema holder though the Data Dictionary or Data Administration tool.

7 ♦ Choose OK. The utility loads the definitions of the P__ files.

The Progress Data Administration tool provides a set of DB2/400 utilities that you can use to maintain a client schema holder. Chapter 9, "Remote Client DB2/400 Utilities," describes these utilities.

Loading Progress Definitions and Data to the AS/400

The next step in moving a Progress database to the AS/400 is to load the data definitions and data. Progress provides utilities that load data definitions (.df) and data (.d) files to the AS/400 server schema.

Follow these steps on your client machine:

1 ♦ Connect to the client schema holder and your Progress/400 server schema.

2 ♦ From the Data Admin main menu, choose DataServer→ DB2/400 Utilities→ Progress/400 Data Dictionary.

3 ♦ Choose the Modify Schema button.

4 ♦ Choose Admin→ Load Data & Definitions→ Load Definitions (.df).

5 ♦ Enter the name of the .df file that you want to load and choose OK.

The utility loads the data definitions into the P__ files in the server schema. In addition, for each table, it creates a physical file on the AS/400. For each index, it creates a logical file.

The Progress length of the combined fields that make up an index must be less than 200 bytes. The Progress/400 length is some number less than 200, depending on the number of key fields used.

6 ♦ Choose Edit→ Commit.

7 ♦ Return to read-only mode.

8 ♦ Choose Admin→ Synchronize Progress/400 Client to synchronize the client schema holder. Select the full synchronization option.

9 ♦ Choose Admin→ Load Data & Definitions→ Table Contents (.d).

10 ♦ Enter the name of the data (.d) file that you want to load.

11 ♦ Repeat until you have loaded all the data files you require.

The utility uses the definitions in the client schema holder to load the data from the .d files into the appropriate data files on the AS/400.

Deploying a Schema Holder

The guidelines and techniques that apply to deploying a Progress database apply to deploying a schema holder for DB2/400 database files. However, you must make changes to **both** the supporting database and the schema holder. For example, when you provide DDS or a SQL script to modify DB2/400 database files, you must also provide a new data definitions file or 4GL code to update the schema holder. Use the Synchronize Progress/400 Client utility to update your deployed schema holder.

3.3.4 Maintaining Data Definitions on the AS/400

The Progress/400 environment is not a static one. It supports two techniques for modifying the DB2/400 data definitions:

- From the client using the Progress/400 Data Dictionary. Use this technique only from Progress clients running in MS-Windows. UNIX clients do not support the Progress/400 Data Dictionary. See Chapter 9, "Remote Client DB2/400 Utilities," for information on using the Progress/400 Data Dictionary.
- On the AS/400 using DDS or SQL. Use this technique if your DB2/400 database files contain data that both Progress and AS/400 high-level languages (HLL) applications access.

When you make changes with DDS or SQL, do the following to make sure that the changes are reflected throughout the Progress/400 environment:

1. Run the CHGPRODCT utility to update the Progress/400 environment with those changes. See Chapter 10, "AS/400 Utilities," for information on using CHGPRODCT.
2. On the client machine, run the Synchronize Progress/400 Client utility to synchronize the client schema holder so that it reflects the changes that you made on the AS/400, or incorporate the `as4sync.p` procedure into your application. You must do this for each client.

3.3.5 Programming Considerations

Before you can access the objects in the Progress/400 environment with a Progress application, you should account for differences in behavior between working with a Progress database and working with DB2/400 database files. For topics to consider see:

- Data types
- ASCII and EBCDIC sort orders
- Locking behavior
- Record creation
- Unsupported Progress statements and functions

See Chapter 2, “Common Product Information,” for a discussion of these topics.

4

Remote Access to Progress/400 DataServer

This chapter describes how the Progress/400 DataServer works. Topics covered in this chapter include:

- How Progress/400 accesses DB2/400 database files
- Remote client connectivity

4.1 How Progress/400 Accesses DB2/400 Database Files

The server schema and the client schema holder allow the DataServer to transmit and process database requests in an n-tier configuration. Once you have set up the server schema and client schema holder, they are transparent to the user and to the application, as Figure 4–1 shows.

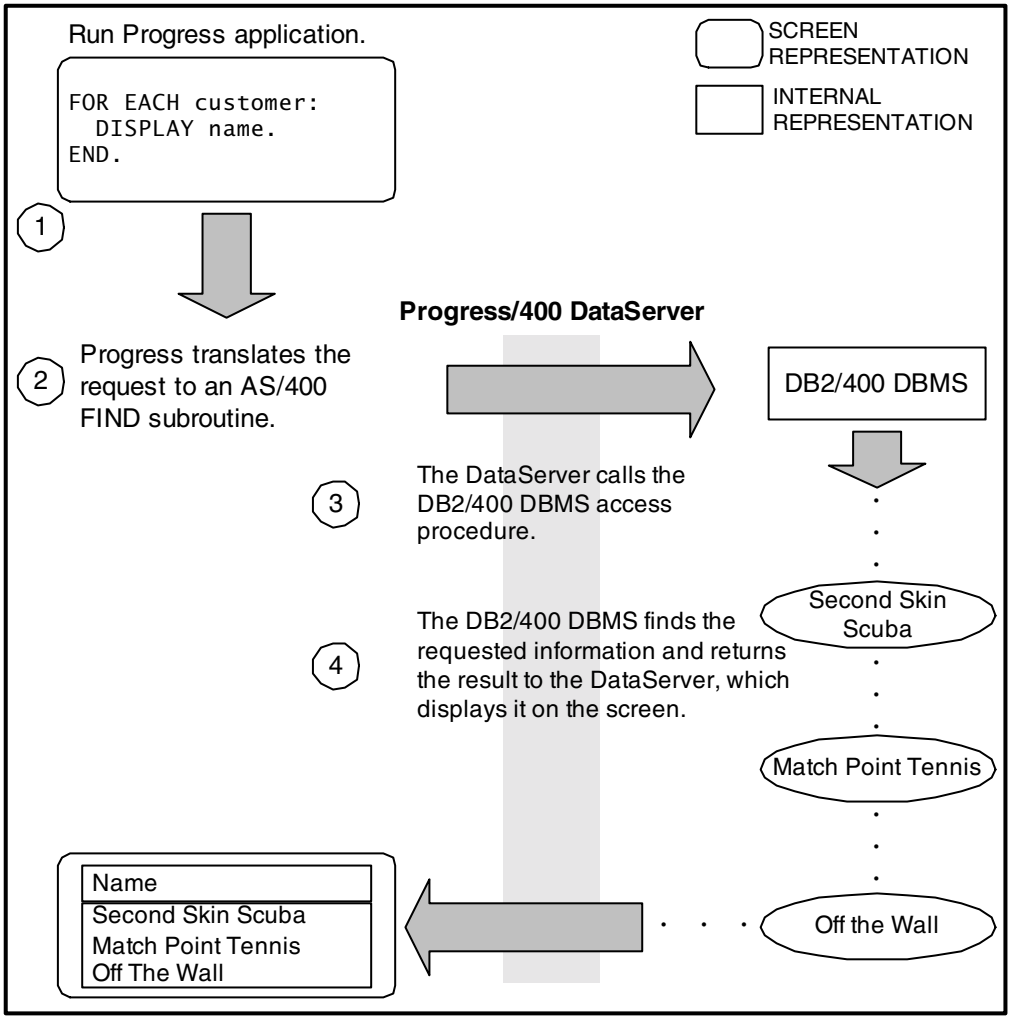


Figure 4–1: How Progress/400 Accesses DB2/400 Database Files

4.2 Remote Client Connectivity

This section describes how to connect the remote Progress client to the AS/400 DataServer. This section also presents information on the architecture, use and management of the TCP/IP and SNA protocols, and specific connection information.

4.2.1 TCP/IP Communications Protocol

The TCP/IP components are:

- TCP/IP broker
- Database server
- Progress client

The TCP/IP broker is a job on the AS/400 that has the capability of starting database servers. It starts one database server per client connection.

Figure 4–2 illustrates how the client and server components interact in a TCP/IP configuration.

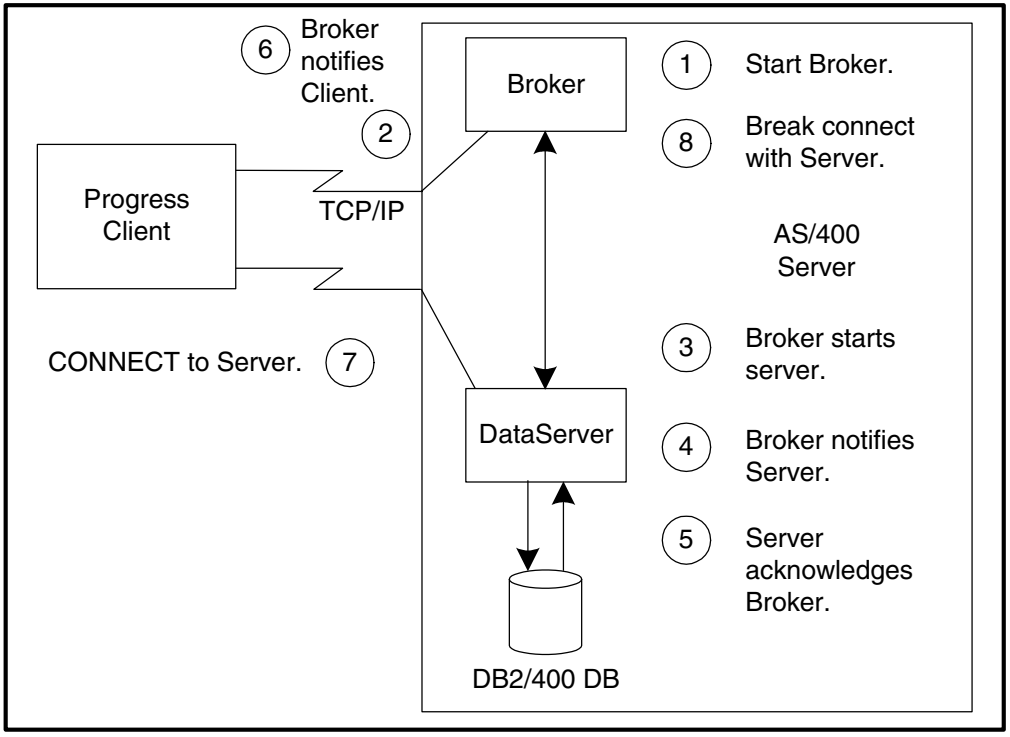


Figure 4–2: TCP/IP Communications Jobs

You must be able to ping the host machine before Progress connects. If you cannot ping the host machine, Progress will not connect.

These notes help to explain Figure 4–2:

1. Start a TCP/IP broker on the AS/400 manually with the `STRPRONET` command. The broker must be started by a user with all object authority (*ALLOBJ).

2. The Progress client issues a CONNECT statement to the server. The CONNECT statement includes information about the network protocol (-N TCP), the host machine (-H), and the service name (-S). The connection is made to the broker that is waiting on the port identified by the service name.

3. When the broker receives the connect information, it starts the DataServer.

4. The broker determines the port that the server will use and notifies the server.

5. The server acknowledges that it received the information about the port.

6. The broker notifies the client of the port that the server will use.

7. The client directly connects to the server using the port number provided by the broker.

8. The broker continues to run as a job on the AS/400 until you manually end it with the ENDPRONET utility.

Because TCP/IP is single threaded, the broker must successfully start one database server job before it can service another. The broker services client requests one at a time only.

Running TCP/IP

You must start the TCP/IP broker with an OS/400 user containing all object authority. However, the actual server jobs run with the level of authority of the client. You specify the client's user profile with the User ID (-U) and Password (-P) connection parameters. A client that uses a profile supplied by IBM (QDOC, QDFTOWN, QPMGR, QPRGOWN, QSECOFR, and so forth) cannot start a server with TCP/IP networking.

Follow these steps to run the TCP/IP broker on the AS/400:

1 ♦ Type **STRPRONET** on the command line, then press **F4** to display the utility's parameters.

2 ♦ Enter the information as described in Chapter 10, "AS/400 Utilities."

To verify that STRPRONET successfully started a broker, do one of the following:

- Check in your AS/400 job log for any messages.
- Run the OS/400 CL command Work with Active Jobs (WRKACTJOB). This command displays the status of the TCP/IP broker job.

  The following sample output is from the WRKACTJOB command:

```
Work with Active Jobs

10:14:26                                                        08/08/95
CPU %:  .0    Elapsed Time:  00:00:00        Active jobs:  12

Opt   Subsystem/Job   User     Type:  CPU %   Function        Status
      PROTCPBRK      QSECOFR  BCH    .0      PGM-PROTCPBRK   SELW
      PROSERVER       KFC     BCH    .0      PGM-PROSERV     TIMW
```

  In this example output, the SELW status for the PROTCPBRK job indicates that the broker is waiting for a client connection request. Note that other states, such as DEQW and TIMW, are normal as long as the job does not remain in those states for extended periods of time.

Managing and Stopping TCP/IP Brokers

To manage a TCP/IP broker, execute the MNGPRONET command on the AS/400. This utility allows you to manage, monitor the status of, or stop a TCP/IP broker. For details, see the "Manage Progress/400 Networking (MNGPRONET)" section in Chapter 10, "AS/400 Utilities."

You can also stop a TCP/IP broker by running the ENDPRONET utility on the AS/400. For details, see the"End Progress Network (ENDPRONET)" section in Chapter 10, "AS/400 Utilities."

4.2.2 SNA Communications Protocol

Connection to the AS/400 through SNA requires SNA routing software on the client machines. You can use the following SNA routers:

- Client Access/400
- Netsoft
- RUMBA

Figure 4–3 illustrates how the client and server components interact in an SNA configuration.

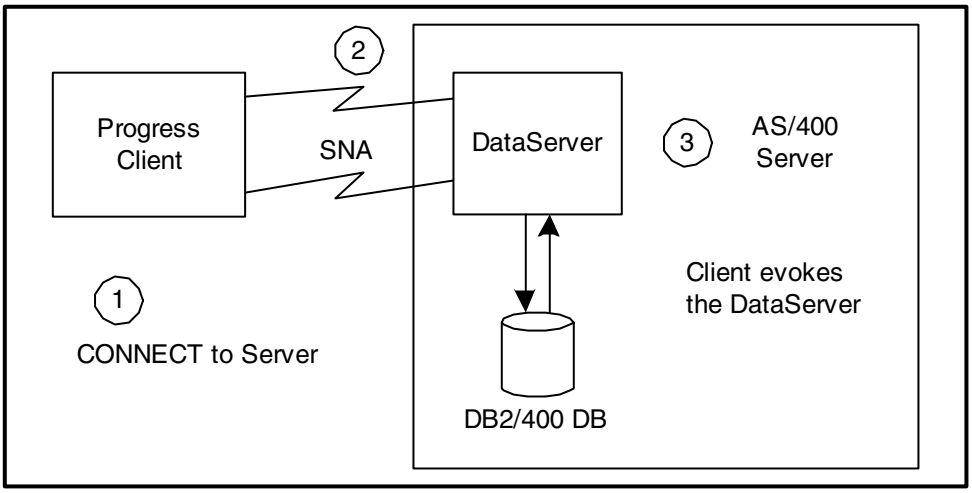


Figure 4–3: SNA Communications Jobs

These notes help to explain Figure 4–3:

1. The Progress client issues a CONNECT statement to the server. The CONNECT statement does not include information about the network protocol, host machine, or user. The SNA routing software provides this connection information.

2. The client connection occurs through the SNA routing software.

3. The client evokes the DataServer job with the authority of the client.

You must be able to establish a 5250 emulation session between the PC client and the AS/400 before connecting Progress. If you cannot establish the emulation, Progress will not connect.

4.2.3 Connection Parameters

Connection parameters control how Progress connects to a database. When using Progress to access a DB2/400 database, use the connection parameters shown in Table 4–1. Be sure to place the parameters that control the connection to a DB2/400 database **after** the database name in a connection statement.

The descriptions of these parameters are specific to connections between Progress clients and the Progress/400 DataServer and the network protocol that your configuration uses.

Table 4–1: Progress/400 Connection Parameters

Parameter	Description	Used By	Protocol
-db	Physical Database Name (Dictionary Library)	All clients	TCP/IP and SNA
-Dsrv	DataServer	All clients	TCP/IP and SNA
-dt as400	Database Type	MS-Windows UNIX	TCP/IP and SNA
-H	Host Name	MS-Windows UNIX	TCP/IP and SNA
-is	Ignore Stamp	All clients	TCP/IP and SNA
-N	Network Type	MS-Windows UNIX	TCP/IP and SNA
-P	Password	MS-Windows UNIX	TCP/IP and SNA
-S	Service Name	MS-Windows UNIX	TCP/IP
-Sn	Server Name	MS-Windows UNIX	SNA
-U	Userid	MS-Windows UNIX	TCP/IP and SNA

NOTE: The -ld parameter is not supported. To change the logical database name, see Chapter 9, “Remote Client DB2/400 Utilities.”

This section and the“DATALIB Argument Usage Notes” section describe the parameters and their usage with the Progress/400 DataServer in detail.

Physical Database Name (-db)

The name of the database that you want to connect to. In the Progress/400 DataServer context, the physical database name is the name of the library that contains the server schema on the AS/400:

SYNTAX

```
-db physical-dbname
```

DataServer (-Dsrv)

Use this parameter to pass connection information directly to the Progress/400 DataServer:

SYNTAX

```
-Dsrv keyword1, value1, keyword2, value2
```

The syntax for specifying the arguments and their values has a second form:

SYNTAX

```
-Dsrv keyword1=value1, keyword2=value2
```

Table 4–2 describes the arguments and values that the -Dsrv parameter accepts. The "DATALIB Argument Usage Notes" section provides some special usage notes on the DATALIB argument.

Table 4–2: DataServer (-Dsrv) Arguments *(1 of 2)*

Keyword	Description
TRANSCTL	Specifies the kind of transaction control that DataServer applications use. There are four options: COMMIT starts commitment control for all the DB2/400 database files associated with a server schema. A journal receiver must exist for each database file to enable transaction management. If you do not have journaling in effect for each DB2/400 file, the application returns an error. LBI specifies that the client use a local before-image file to manage main transactions and nested subtransactions. See the "Transaction Control" section in Chapter 8, "System Administration," for more information. NONE specifies that there is no transaction control in effect. OPTIONAL specifies that the DataServer use commitment control for all DB2/400 files for which there is an active journal. This allows the use of transaction management for some files and not for others. The DataServer opens files for which there is no active journal without commitment control. This is the default.
COMPRESS	Specifies that the client and the DataServer use compression when transferring database records back and forth. Compression also allows the DataServer to transfer more records to the client in each message when using pre-fetch. The allowed values are: 0 = off (the default) 1 = on
SBS	Specifies that the DataServer uses selection-by-server when appropriate. The allowed values are: 0 = off 1 = on (the default)
DATALIB	Specifies the library in which the DataServer opens DB2/400 files. If you specify a library name, you must enter it in capital letters. To open files using the DataServer's library list, specify the value *LIBL instead of a library name. The default is to open files in the library whose name is stored in the P__FILE server schema file. See the "DATALIB Argument Usage Notes" section for more information.

Table 4–2: DataServer (-Dsrv) Arguments *(2 of 2)*

Keyword	Description
LFLVLCHK	Enables the CRC method of logical file level checking. In this method, the CRC is created from the file's name, record format, key field names, types, and lengths to ensure that its CRC does not change if the file is moved or copied. CRC = on. The default is off.
CANCELQRY	For TCP/IP connections, allows the client to use **CTRL-C** (on UNIX machines) or **CTRL-BREAK** (on Windows machines) to cancel a query request. By default, control does not return to the client until the query completes. The allowed values are: 0 = off (the default) 1 = on This argument is not available for SNA connections.

4.2.4 DATALIB Argument Usage Notes

The DATALIB argument to the -Dsrv parameter controls how the DataServer opens DB2/400 files. By default, DB2/400 files are opened in the library specified by the library name stored in the P__FILE server schema file. Progress/400 sets up this library name when you create a file using the Progress/400 Data Dictionary or when the `CHGPRODCT` command places the file definition into the server schema. The DATALIB connection parameter expands how the DataServer locates files.

When using this argument with the CONNECT statement, use the following syntax:

SYNTAX

```
CONNECT -db DCTLIB -Dsrv DATALIB=VALUE
```

You must enter *VALUE* in capital letters. You can specify either an explicit library name or the value *LIBL. Do not insert spaces before or after the equal sign (=).

Suppose, for example, that you connect without specifying an explicit library name, and the library defined in the P__FILE is named TEST. You now execute the following CHGPRODCT command, which executes for the Dictionary Library named DCTLIB:

```
CHGPRODCT PRODCT(DCTLIB) FRMFILLST(((*ALL) TEST))
```

This CHGPRODCT command places all valid file definitions from the library TEST into the server schema of the Dictionary Library DCTLIB. Now suppose that you instead connect as shown in the program PGM1.P:

PGM1.P

```
CONNECT SCHLHD -1.
CONNECT DCTLIB -Dsrv DATALIB=PROD.

RUN PGM2.P
```

The DataServer now opens all files during this connection in the library PROD rather than in the library TEST.

The following example illustrates the use of the DATALIB value *LIBL. Suppose that you connect as shown in the program PGMA1.P:

PGMA1.P

```
CONNECT SCHLHD -1.
CONNECT DCTLIB -Dsrv DATALIB=*LIBL.

/* Establish the library list for the server */

CREATE QCMD.
ASSIGN CMD = "* CHGLIBL (QTEMP QGPL PROGRESS CUSTLIB TEST PROD)".
VALIDATE QCMD.
RUN PGM2.P
```

Because you specified *LIBL as the DATALIB value, the DataServer opens files during the current connection based on the DataServer's library list. Now suppose that you execute the following query:

```
FOR EACH customer where cust-num > 50:
VALIDATE QCMD.
RUN PGM2.P
```

Assuming that the CUSTOMER file exists in library CUSTLIB, it is opened for this library. Notice how the library list was set up in PGMA1.P before any files are opened. This is important: if a file opens before the library list is established, the DataServer might not open the desired file. By specifying *LIBL as the DATALIB, the DataServer performs file opens based on library list, just as RPG does.

CAUTION: **Do not** use the Ignore Stamp (-is) switch when using -Dsrv DATALIB=*VALUE*. Progress does not perform file-level checking when the DataServer connects using the -is switch, and if you specify DATALIB in conjunction with the -is switch, it might open an incorrect file. For example, in the example program PGM1.P (shown earlier in this section), suppose that the library TEST contains a CUSTOMER file for the AP application and the library PROD contains a CUSTOMER file for the distribution application, and that each of these files has a different record format. Because Progress does not perform file-level checking, the DataServer might open a file in a different library than the one you specified with CHGPRODCT, resulting in data corruption.

Note that the DATALIB argument does not work with virtual tables. If you specify this argument, logical files that Progress/400 treats as a virtual file cannot be found if they were not in the Dictionary Library, specified when you created the server schema. In this case, the following error messages appear:

- On the client: “Unable to open the database file. (2468)”
- In the server job log: “PRO9024-There is a server schema/object mismatch for *file*.”

Database Type (-dt)

The type of non-Progress data source that you want to connect to. The default is as400. This parameter is optional:

SYNTAX

```
-dt type
```

Host Name/Profile Name (-H)

For Windows clients using SNA, the Host Name (-H) parameter specifies the Partner LU name. The default for Windows is the AS/400 host name from the CONFIG.PCS file.

For Windows clients using Rumba, use the -H parameter to specify the LU Alias, the PLU, and the Mode name. If you are connecting to multiple AS/400s, you must supply this parameter for each AS/400. The default is:

PLU=5250PLU/LU=5250LU/MODE=QPCSUPP

SYNTAX - Windows

```
-H PLU=plu-name/LU=lu-name/MODE=mode-name
```

For clients using TCP/IP, the Host Name (-H) parameter specifies either the name of the AS/400 in the host file or that you should use the IP address of the host.

Ignore Stamp (-is)

When accessing DB2/400 database files, the DataServer validates the file definitions in the schema holder against the current state of the file. Within a given Progress session, whenever you access a file or index for the first time, Progress/400 compares CRCs stored in the server schema for that file or index with the actual CRCs of the AS/400 physical file. If the CRCs do not match, Progress returns an error and denies you access to the data.

This verification takes time. In a production environment, if you are certain that the data definitions in the DB2/400 database files that you are accessing have not changed, you might want to consider using the -is parameter to bypass CRC and level checking. It is very important to understand that if you use the -is parameter and the DB2/400 files have changed, the data can be corrupted:

SYNTAX

```
-is
```

NOTE: The OS/400 data dictionary time stamp changes whenever you make **any** change to the DB2/400 database structure, regardless of whether or not you save the change.

Network Type (-N)

Use this parameter to specify the networking protocol:

SYNTAX

```
-N network-type
```

Table 4–3 lists the network types that the Progress/400 DataServer supports and the values that you specify with -N.

Table 4–3: Network Type (-N) Arguments

Network Type	Value
SNA	as400sna
TCP/IP	TCP

User ID (-U)

Use this parameter to specify the user ID. The userid value varies depending on the client type.

This parameter is required for connection by all remote clients. These connections also require the Password (-P) parameter:

SYNTAX

```
-U userid
```

Password (-P)

The Password (-P) parameter specifies the AS/400 password of the user starting the Progress/400 session. You must capitalize the password, and you must use this parameter with the User ID (-U) parameter.

The password that you specify is checked against AS/400 user profiles. If it is valid for an AS/400 user, OS/400 starts the Progress/400 DataServer job on the AS/400. The job runs under the attributes of the associated user profile. If the password is not valid for the AS/400, or if the user profile does not have authority to the dictionary library specified in the connect, the connection fails:

SYNTAX

```
-P Password
```

Service Name (-S)

Use this parameter for a TCP/IP connection. It specifies the name of the service on the AS/400 that you called when you started the broker. Select the service name from the TCP/IP services file on your client machine.

Server Name (-Sn)

Use this parameter for an SNA connection. It specifies the server program started on the AS/400 by a Progress connection request. This is an optional connection parameter:

SYNTAX

```
-Sn server-name.prolibraryname
```

Table 4–4 describes the values for the -Sn parameter.

Table 4–4: Server Name (-Sn) Options

Server Name	Description
prosna.*prolibraryname*	Starts a Progress server on the AS/400. It saves a job log only when the server process ends abnormally with an AS/400 severity level of 20 or higher. The value prosna is the default for *server-name* and *LIBL is the default for *prolibraryname*.
prosnal.*prolibraryname*	Starts the Progress server on the AS/400. This option creates and saves a job log each time you successfully connect to the AS/400. *LIBL is the default for *prolibraryname*.

Note that you can substitute the consna and consnal programs for the prosna and prosnal programs. However, using prosna and prosnal allows you to customize how the Progress/400 DataServer is started.

4.2.5 Connection Techniques

The various Progress clients that access DB2/400 databases use different techniques to communicate with the AS/400. As a result, some connection parameter requirements differ among clients:

- From the command line at startup
- Using the CONNECT statement from within a Progress session
- Using the auto-connect feature from within Progress applications
- Using the Progress/400 Data Dictionary tool or the Data Administration tool

The following sections provide examples for connecting to DB2/400 databases in the supported configurations. Regardless of the connection technique that you choose, you must first connect

to the schema holder, then connect to the DB2/400 database. For example, if you choose to connect at startup, specify the schema holder and associated connection parameters **before** you specify the DB2/400 database and its associated connection parameters.

See the database access chapter of the *Progress Programming Handbook* for more information about connecting to a database and for details on choosing a connection technique.

Connecting at Startup

You can connect to both a schema holder and DB2/400 database files when you start a Progress client session. The following examples illustrate how to connect a Progress server for a schema holder for multiple users. If you want to connect to a schema holder in single-user mode, use the Single-user (-1) parameter.

NOTE: If you prefer to prompt the user for a user ID and password, see the "Application Security" section in Chapter 2, "Common Product Information."

Windows Clients

This Windows example connects to a Progress server for a schema holder called abc in the TEST subdirectory and a DB2/400 database called xyz:

```
_PROWIN.EXE C:\TEST\abc -1 -db xyz -U username -P password
```

You can also include this line in a Windows program item definition to connect to your schema holder automatically.

UNIX Clients

This example shows a connection to a Progress server for a schema holder called abc and a DB2/400 database called xyz. The logical connection profile name is profile, the AS/400 user ID is *username*, and the password is *password*:

```
pro abc -db xyz -H profile -U username -P password
```

TCP/IP Connection Using the CONNECT Statement

Use the CONNECT statement to connect to a DB2/400 database from each of the Progress client types: Windows and UNIX. The following sample CONNECT statement is valid for Windows and UNIX clients. The example assumes that the Progress session was started by connecting to a schema holder for the DB2/400 database named xyz:

```
CONNECT xyz -H hostname -N TCP -S servicename -U username -P password
```

SNA Connection Using the CONNECT Statement

Use the CONNECT statement to connect to a DB2/400 database from the Windows client. The example below assumes that the Progress session was started by connecting to a schema holder for the DB2/400 database named xyz:

```
CONNECT xyz -U username -P password -N as400sna -Sn servername -H hostname
```

SNA and TCP/IP Connection Using Auto-connect

The auto-connect feature allows you to connect to databases automatically as required during program execution. To perform an auto-connect operation, Progress uses information stored in a schema holder to connect to a second application database. It does this before running compiled procedures that access the second database. The primary application database (that contains the auto-connect information) must be connected before Progress can perform an auto-connect.

4.2.6 General Connection Considerations

When connecting to databases, there are three issues to consider:

- Logical database names
- Connection modes
- Message buffer size

Logical Database Names

A *logical database name* is a database reference representing the name of a connected physical database. The logical database name is used to resolve database references. When a procedure is compiled against a database, Progress stores the logical database name in the procedure's object code. When you execute a procedure, the databases referenced by the procedure must match the logical name of a connected database.

When you connect to a database, it is automatically assigned a default logical name for the session. The default logical name is the physical database name, which is the OS/400 library name.

A database connection fails if the logical database name of the database that you are connecting to is the same as the logical name of an already connected database. If you try to connect a database with a logical database name that matches the logical database name of an existing, connected database of the same database type (Progress, DB2/400, ORACLE, and so forth), Progress assumes that the database is connected and ignores that request.

You can change a database's logical name from a Progress procedure either with the CREATE ALIAS statement or by using the Edit Connection Information utility in the Data Administration tool. See Chapter 9, "Remote Client DB2/400 Utilities," for more information.

Connection Modes

You can make the connection to the schema holder in single-user or multi-user mode. You are always connected to DB2/400 databases in multi-user mode. Single-user mode does not prevent others from accessing the DB2/400 database:

- Other Progress users and other AS/400 users might be accessing the DB2/400 database using a non-Progress language (for example, COBOL or RPG).
- Other Progress users might be accessing the Progress/400 database using a different (but similar) schema holder.

Using the Message Buffer Size (-Mm) Startup Parameter

This section provides some special usage notes on the Message Buffer Size startup parameter. This parameter instructs the Progress Client and the Progress/400 DataServer to create a buffer of a specified number of bytes. When a NO-LOCK query requests records, the DataServer packs records into this buffer until it contains all of the records that it can hold. At this point, the DataServer sends it to the Client in one transfer. Note that the -Mm startup parameter affects **only** Progress, it does not influence the hardware packet sizes.

Suppose, for example, that the hardware packet size on your network is 1500 bytes and the -Mm startup parameter is set to 4000 bytes. The DataServer sends a packet of 4000 bytes to the hardware. The hardware breaks the packet and sends 1500 bytes at a time over the communications line. On the other side of the line, the hardware reassembles these packets into the actual Progress buffer.

Because the hardware handles breaking and reassembling the buffer so rapidly, you should use the largest value possible for the -Mm startup parameter. The maximum size for the message buffer is 4094 on SNA. Using these larger message buffer sizes typically improves the overall performance of your Progress/400 queries in a networked environment.

One particular Progress error message might occur during connection to the Progress/400 DataServer. The following message might be returned when the AS/400 dictionary library name cannot be found (for example, if it is misspelled or does not exist):

```
Server has -Mm parm -200 and client has 1024. They must match. (1150)
```

Check your connect parameters to make sure that they are not the cause of this message.

4.2.7 Connection Troubleshooting

Table 4–5 describes circumstances that can produce connection failures and how Progress responds to them.

Table 4–5: Connection Failure Responses

Failure Circumstance	Progress Response
During startup	Progress displays an error message and returns to the operating system prompt.
During a CONNECT statement	Progress terminates the remainder of the CONNECT statement and a run-time error occurs. Connections made previous to the failed connection remain in effect. Use the NO-ERROR modifier with the CONNECT statement to trap run-time errors. If you use the NO-ERROR modifier and it fails, you see the same failure behavior as with an unsuccessful CONNECT statement. However, run-time errors do not occur. Use the CONNECT statement to test for a valid connection.
During an auto-connect	Progress terminates the remainder of the auto-connect and a run-time error occurs. Any connections made previous to the failed connection remain in effect. You cannot trap auto-connect run-time errors.
During an attempt to connect using the Progress Data Dictionary	The Progress Data Dictionary displays an error message.
During an attempt to connect to a connected database with the same logical name	Progress responds with a warning. You can continue. (Use NO-ERROR to suppress the warning.)
During an attempt to connect to an unconnected database when the logical name is in use by a connected database	Progress responds with a run-time error. You cannot connect to the second database.

NOTE: Use the CONNECTED function to see if you are already connected. For more information, see the *Progress Language Reference*.

There are several reasons that a connection attempt to a Progress/400 database can fail:

- The Progress schema holder is not connected.
- The user is not authorized to access the OS/400 library (the database).
- The userid and password combination provided during connection is invalid.
- The wrong server schema library version is being used.

When the server terminates unexpectedly, the Progress client is sometimes left in a semiconnected state. When the client-server connection is lost unexpectedly, make sure to issue a DISCONNECT statement before you attempt to reconnect.

NOTE: If the server terminates unexpectedly without producing a job log, do the following to ensure that the system generates a job log:

- For SNA connections, specify the connection parameter -Sn prosnal.PROGRESS at startup.
- For TCP/IP connections, set the STRPRONET server logging parameter when you start the TCP/IP broker.

Use the resulting job log to help diagnose the connection problem.

Table 4–6 describes common error messages that you might encounter during connection and suggests responses.

Table 4–6: Connection Failure Error Messages *(1 of 2)*

Error Message	Response
You cannot use AS/400 communications for the *dbname* database.	1. Verify that the database name is spelled correctly. 2. Verify that the schema holder is connected and describes the specified database.
Partner LU alias or mode name unknown.	Check the PLU alias name and the mode name.
User ID/Password combination not accepted by AS/400.	Check the case-sensitive user ID and password passed by the -U and -P connection parameters.
The name of the server program was not recognized.	Verify that Progress was correctly installed.

Table 4–6: Connection Failure Error Messages *(2 of 2)*

Error Message	Response
The server program appended on the AS/400.	Check the OS/400 job log for information about the server program.[1]
The -H parameter for AS400SNA should be [LU=.../PLU=.../MODE=...].	Check the case-sensitive -H connection parameter.
The -H parameter LU, PLU, and MODE values cannot exceed 8 characters.	Check the case-sensitive -H connection parameter.
Could not establish initial conversation correctly.	Check the client job log, the OS/400 job log, or the operator's message queue.
Unspecified error during initial send to AS/400.	Check the client schema job log or the OS/400 job log.
Unspecified error during initial receive from AS/400.	Check the client schema job log or the OS/400 job log.
Unspecified allocation error (AS/400 side).	Check the client schema job log or the OS/400 job log.
Unspecified error trying to initialize the session.	Verify basic connectivity between the client and the AS/400. If you are connecting through SNA, check that you can establish a 5250 session. If you are connecting through TCP/IP, use the `ping` command to check the connection.
Unspecified parameter error trying to allocate the session.	Verify basic connectivity between the client and the AS/400. If you are connecting through SNA, check that you can establish a 5250 session. If you are connecting through TCP/IP, use the `ping` command to check the connection.

Disconnecting from DB2/400 Databases

To disconnect from your target database, do any of the following:

- Use the DISCONNECT statement from your Progress session.
- Use the QUIT statement from your Progress session.
- Disconnect through the Progress/400 Data Dictionary tool.
- Disconnect through the Data Administration tool.

5

Preparing to Use AS/400-based Clients

This chapter explains how to set up and use the Native 4GL Client and the Progress/400 AppServer. This chapter provides the following information:

- Understanding the Integrated File System
- A task list
- Creating a schema image
- Executing Progress code from the native clients
- Internationalizing the native clients

5.1 Overview

There are two AS/400-based clients that are discussed in this chapter, the Native 4GL Client and the Progress/400 AppServer. In relation to the Progress/400 DataServer, they are both considered clients. The phrase native client is used to refer to both clients on the AS/400.

5.2 Understanding the Integrated File System

The Integrated File System (IFS) is an umbrella file system for all of the OS/400 file systems. Users can access any OS/400 file system through the IFS. The Progress/400 environment uses the IFS to manage both p-code file transfer from a remote client and Progress application execution.

The IFS allows the native clients to support either UNIX-style paths (/) or Windows-style paths (\). Progress Software Corporation recommends that you use UNIX-style paths (/) when specifying paths in the IFS. Even though IFS allows either forward or back slashes, Progress does not allow mixing them. Since the client products are installed in a directory on the AS/400 specified with the forward slash, you should use the forward slash to avoid confusion.

For a detailed description of the IFS, see IBM's *Integrated File System Introduction*. For information on specific AS/400 versions that support IFS features, see the *Progress/400 Release Notes.*

5.2.1 The ROOT File System Under the IFS

One specific file system, ROOT, can be accessed only through the IFS. The ROOT file system is POSIX compliant in that files are stored in a hierarchy of directories where stream files reside. The ROOT file system is ideal for managing stream files, such as p-code. Unlike database files, stream files are not broken down into individual records with fixed sizes. Instead, stream files can be variable length and have different data formats. EBCDIC stream files might be text files such as p-code. Binary stream files might be graphics or object code, such as r-code.

Remote clients, such as a Windows client, store p-code as a stream file. You can transfer this p-code to the AS/400 where the native clients then execute the code. P-code that contains embedded UNIX-style or Windows-style paths can execute on the AS/400 through the ROOT file system.

The ROOT file system top-level directory is denoted by the slash (/). With the ROOT file system, you can denote a path to access stream files. The path can be a hierarchy of directories pointing to where a file exists. For example, a path might be made up of the following:

```
/directory1/directory2/object
```

Under directory1 is another directory called directory2 that contains an object. In Progress/400, this object might be p-code or r-code.

Your home directory resides under the top level of the ROOT file system. In the ROOT file system, your home directory would be /home. You can use directories underneath the home directory as one place to store stream files.

5.2.2 Absolute and Relative Paths

Absolute paths always begin the search from your root directory. Relative paths begin the search from the current directory. With the absolute path, the root directory is denoted by the slash (/). For example, suppose that your path looks like the following:

```
/directory1/directory2/object
```

In this case, the ROOT file system searches from the following top level:

```
/directory1/directory2/object
```

However, if you do not specify the first slash (/), you have a relative path. When you use a relative path, Progress/400 starts with your current working directory. For example, suppose that your path looks like the following:

```
directory1/directory2/object
```

In this case, the ROOT file system searches from the following:

current-working-directory`/directory1/directory2/object`

5.2.3 Accessing QSYS.LIB Files

To access objects in the QSYS.LIB file system, such as p-code, you embed your QSYS.LIB file system path in IFS path syntax. Although Progress/400 supports QSYS.LIB, it is recommended that you store your stream files in the root file system. Table 5–1 shows how to specify your QSYS.LIB file system path in a path within the IFS file system.

Table 5–1: QSYS.LIB and IFS Path Resolutions

QSYS.LIB Path Resolution	IFS Path Resolution
*LIBL/REPORT(REPORT)	/QSYS.LIB/*LIBL.LIB/REPORT.FILE/REPORT.MBR
*CURLIB/MKTG(REPORT)	/QSYS.LIB/*CURLIB.LIB/MKTG.FILE/REPORT.MBR
DEMO/MKTG(REPORT)	/QSYS.LIB/DEMO.LIB/MKTG.FILE/REPORT.MBR

Progress/400 native clients search for p-code and r-code in the ROOT file system within IFS similarly to the Progress 4GL. However, if you store the p-code and r-code in the QSYS.LIB directory structure, you must fully qualify where the p-code or r-code resides when you use this

directory structure in your Progress code. If you do not fully qualify the path, the native clients perform no search of any kind.

For example, if the `abc` procedure is in the ROOT file system (IFS), the native clients look for `abc.r` and then `abc.p`. However, if the procedure is in the QSYS.LIB file system, your RUN statement must be fully qualified; for example:

```
RUN /QSYS.LIB/*LIBL.LIB/P.FILE/ABC.MBR
RUN /QSYS.LIB/*LIBL.LIB/R.FILE/ABC.MBR
```

For more information on how Progress/400 uses paths, see the "Executing Progress Code from the Native Clients" section in this chapter.

5.3 Task List

Follow these steps to use the native clients:

1 ♦ Create a dictionary library containing the schema image on the AS/400. You create the dictionary library and the schema image with the DUPPRODB utility if you do not have an existing server schema, or with CRTSCHIMG if you have a server schema.

2 ♦ Prepare your Progress 4GL code for the AS/400. This might involve removing user interface statements and redirecting input and output.

3 ♦ Transfer your p-code to the AS/400 from a remote client, or locate local AS/400 p-code or r-code. If you are transferring p-code from a remote client to the AS/400, use the Progress 4GL to check the syntax before you execute the transfer.

4 ♦ Compile your p-code using the Native 4GL Compiler on the AS/400. Any compilation errors show up in your job log. You can fix these compilation errors on the remote client where you can use the Procedure Editor.

5 ♦ Invoke a client to execute your Progress 4GL code (p-code or r-code).

5.4 Creating a Schema Image

Create a schema image by creating a new Progress/400 environment or by modifying an existing one, as the "New DB2/400 Database" section describes.

5.4.1 New DB2/400 Database

To set up a new Progress/400 environment, execute the DUPPRODB utility with the Create Schema Image (CRTSCHIMG) parameter set to *YES. The CRTSCHIMG parameter adds the P__SCHEMA* files to the server schema, then synchronizes it with the server schema's P__* files. The P__SCHEMA* files contain schema definitions for the DB2/400 database. The schema is a description of the DB2/400 database structure, the files, the fields within the files, the indexes, and so forth. The Progress/400 DataServer uses the schema image to map the data definitions in the server schema to a format the native clients can understand.

Use the following OS/400 syntax to execute the DUPPRODB utility with the CRTSCHIMG option:

```
DUPPRODB NEWDB(library-name) CRTSCHIMG(*YES)
```

Table 5–2 details the Create Schema Image (CRTSCHIMG) parameter for the DUPPRODB utility. For a complete list of the DUPPRODB parameters, see Chapter 10, "AS/400 Utilities."

Table 5–2: DUPPRODB Parameter

Parameter	Keyword	Value
Create Schema Image	CRTSCHIMG	Creates a schema image for the native clients. CRTSCHIMG populates the server schema with P__SCHEMA* and automatically synchronizes. The default value for this parameter is *NO.

5.4.2 Existing Server Schema

If you have an existing server schema, execute the CRTSCHIMG utility to add a schema image. CRTSCHIMG adds the P__SCHEMA* files to the server schema, then synchronizes it with the server schema's P__* files. The P__SCHEMA* files contain schema definitions for the DB2/400 database. The schema is a description of a DB2/400 database structure, the files, the fields within the files, the indexes, and so forth.

Use the following OS/400 syntax to execute the CRTSCHIMG utility:

```
CRTSCHIMG DCTLIB(library-name)
```

5.5 Executing Progress Code from the Native Clients

With the native clients, you can make large database updates, generate reports, or perform other host-based operations that free up the remote client and reduce network traffic. The native clients do not provide interactive display capabilities. They take input from a file and direct output to a file or a printer.

5.5.1 Preparing Progress 4GL Procedures for the AS/400

This section provides guidelines for preparing 4GL code for use on the AS/400. Topics covered include removing user-interface statements, using aliases, and file input and output.

Removing User-interface Statements

Because the native clients do not provide any screen-oriented display capabilities, you must make sure that your application output is redirected to a file or printer with the OUTPUT TO statement. Use the remote client's Procedure Editor to edit your p-code and redirect any output. See Chapter 2, "Common Product Information," for additional guidelines.

Using Aliases

When using DEFINE ALIAS so that r-code can be run against multiple databases, you must define two aliases, as follows:

- An alias for the database, which is the library name
- An alias for the schema image whose name is library-name-SCHEMA

File Input and Output

Applications must use either the QSYS.LIB file system or IFS file system syntax when performing file input and output, as the following sections describe.

INPUT FROM and OUTPUT TO Commands

Because the native clients do not have interactive capabilities, you must explicitly define your input and output streams with Progress statements. You cannot default to the terminal for input and output as you would with an interactive Progress procedure. The INPUT FROM and OUTPUT TO commands in your native client applications behave the same as the standard Progress 4GL with the exception of the interactive capability.

To provide a location for the input and output that you generated with the client code on the AS/400, you must indicate a file using INPUT FROM and OUTPUT TO. The syntax that you use when specifying a file depends on the file system that you are using on the AS/400.

Table 5–3 gives examples of the syntax for INPUT FROM in the QSYS.LIB file system or the ROOT file system under IFS.

Table 5–3: Integrated File System INPUT FROM Syntax

File System	Syntax
QSYS.LIB	INPUT FROM '/QSYS.LIB/**LIBL*.LIB/*P*.FILE/*MYFILE*.MBR'
QSYS.LIB	INPUT FROM '/QSYS.LIB/**CURLIB*.LIB/*P*.FILE/*MYFILE*.MBR'
QSYS.LIB	INPUT FROM '/QSYS.LIB/*MYLIB*.LIB/*P*.FILE/*MYFILE*.MBR'
ROOT	INPUT FROM '*/DIRECTORY1/DIRECTORY2/MYFILE.P*'
ROOT	INPUT FROM '*DIRECTORY1/DIRECTORY2/MYFILE.P*'
ROOT	INPUT FROM '*MYFILE.P*'

When you specify INPUT FROM '/QSYS.LIB/**LIBL*.LIB/*P*.FILE/*MYFILE*.MBR', Progress/400 searches for the file/member in the current library list. If multiple files of the same name exist in different libraries, Progress/400 uses the first file it finds. If the file is not found in the library list, Progress/400 writes an error to your job log.

It is important to note that the INPUT FROM command does not use PROPATH to search for files. Progress/400 does, however, search the **LIBL* and **CURLIB* for a file when you specify these library lists.

Table 5–4 gives examples of the syntax for OUTPUT TO in the QSYS.LIB and ROOT file system under the IFS.

Table 5–4: Integrated File System OUTPUT TO Syntax

File System	Syntax
QSYS.LIB	OUTPUT TO '/QSYS.LIB/**LIBL*.LIB/*P*.FILE/*MYFILE*.MBR'
QSYS.LIB	OUTPUT TO '/QSYS.LIB/**CURLIB*.LIB/*P*.FILE/*MYFILE*.MBR'
QSYS.LIB	OUTPUT TO '/QSYS.LIB/*MYLIB*.LIB/*P*.FILE/*MYFILE*.MBR'
ROOT	OUTPUT TO '*/DIRECTORY1/DIRECTORY2/MYFILE.P*'
ROOT	OUTPUT TO '*DIRECTORY1/DIRECTORY2/MYFILE.P*'
ROOT	OUTPUT TO '*MYFILE.P*'

OUTPUT TO Rules

The following list describes the behavior of the OUTPUT TO command:

- ****CURLIB*.lib —** With the OUTPUT TO statement, if you specify OUTPUT TO '/QSYS.LIB/**CURLIB*.LIB/*P*.FILE/*MYFILE*.MBR', Progress/400 writes *MYFILE*.MBR to wherever **CURLIB* is defined. If a current library is not set in the library list for the job, or if the current library is set but the file does not exist, Progress/400 writes an error to the job log.

- **Library explicitly stated —** With the OUTPUT TO statement, where a library, file, or member is explicitly stated but does not exist, Progress/400 creates a library and file as a source physical file with a record length of 144, and a member within the source file.

- **End of record —** The OUTPUT TO statement uses stream input and output to write data for both ROOT and QSYS.LIB file systems. Since stream files do not use end-of-record, QSYS.LIB files are not truncated at the end of a record. Instead, the output wraps into the next record.

- **Stream file format —** If you specify a physical file member in the QSYS.LIB library file system as the destination of an OUTPUT TO statement, the formatting can be confusing. Data written to this file is output in stream file format. You must convert your data to a record format if you want to view it with the QSYS.LIB file system.

- **Record file format —**To convert stream data to the record-oriented format of the QSYS.LIB file system, follow these steps:

  a) Execute the CPYFRMSTMF command. The file you specified on the OUTPUT TO statement in your 4GL maps to the FROMSTMF parameter of the command.

  b) Select your desired record file member name in the TOMBR parameter. You must also select the *LF value for the ENDLINFMT parameter {ENDLINFMT(*LF)}.

  NOTE: Using command defaults for the ENDLINFMT parameter can result in extra carriage returns and line feeds that fragment the data. This is the default behavior in IFS when the FROMSTMF parameter specifies a file from the QSYS.LIB file structure. The ENDLINFMT selection is not necessary when FROMSTMF parameter is from the IFS file structure instead.

Output to Printer

When you direct output to a printer, OS/400 writes the output to a spooled file and places it in an output queue. The characteristics of the spooled file are controlled by the AS/400 printer file. The AS/400 printer file determines such things as the page size and the number of lines per page for printed output. Progress/400 supports a printer file called PROPRTF, which resides in the Progress/400 executables library. The PROPRTF printer file uses most of the OS/400 command defaults for print files including the default page size, font, page rotation, and number of copies. It also uses the default printer device and output queue associated with the job that executes the Progress procedure. The PROPRTF printer file differs from the AS/400 printer file defaults for the following attributes:

```
RPLUNPRT(*YES) CLTCHAR(*FCPC) CHLVAL((1(1)) (2(57)))
```

You can override the default characteristics of the PROPRTF print file, or direct the output to a different printer by using the OVRPRTF command. If you do override the printer default, you must make sure the PRTF attributes are set for the new printer.

Additionally, you can direct output to a specific printer file by providing the printer name as an argument to the OUTPUT TO statement. Any printer file you specify must contain the attributes assigned in the PRTF printer file.

Note that if you output to a printer using PUT UNFORMATTED, unpredictable results can occur. The Progress 4GL does not have any knowledge of the AS/400 print control. The 4GL programmer must manage print control.

Another option for specifying your own printer file is using the -o parameter. The Progress statement OUTPUT TO PRINTER automatically uses the correct attributes for you when you use the Printer (-o) parameter.

NOTE: Blank lines from your print file are not visible when viewed from an output queue. Printed output contains the proper spacing.

5.5.2 Transferring Progress 4GL Procedures to the AS/400

This section provides guidelines for transferring 4GL code to the AS/400:

- Remove all references to terminal input or output. Use the INPUT FROM and OUTPUT TO statements.
- Select a method to transfer code to the AS/400. Options for transfer might be using FTP, Client Access, or Rumba. If you do not have any of these applications, Progress provides a sample procedure that might help. Refer to your Progress/400 Release Notes for more information.
- Transfer the code. Once transferred, the code resides in the IFS file system.

Writing Your Own Transfer Program

To transfer files to the AS/400 using the Progress 4GL, use the QCMD interface. For a definition of the QCMD interface, see Chapter 11, "Progress 4GL Interfaces to OS/400 Languages and Objects." Progress/400 provides three server commands that specifically handle file transfer. Table 5–5 describes these commands.

Table 5–5: QCMD Commands for File Transfer

Command	Description
OPNSTMF	Opens an OS/400 stream file
WRTSTMF	Writes to an OS/400 stream file
CLOSTMF	Closes an OS/400 stream file

Follow these steps to transfer p-code to the AS/400:

1 ♦ Develop your code on your remote client. Make sure to remove user-interface syntax.

2 ♦ Perform a syntax check on the remote client to make sure the p-code is valid.

3 ♦ Transfer the code to the AS/400 from the remote client.

5.5.3 PROPATH Construction Rules

PROPATH on the OS/400 is built like PROPATH on all other Progress clients. However, you can lose the working directory in the PROPATH if you do not specify enough commas in the PROPATH sections you define, as prepending the sections can alter the PROPATH meaning. For example, if you specify the string `/home/mydir` as the PROPATH, it is prepended to the default PROPATH to produce the following:

```
/home/mydir,/dlc
```

Note that the single comma acts as a path delimiter and does not itself represent the working directory. To insert the working directory into the middle of the PROPATH, specify the following for PROPATH:

```
/home/mydir,
```

This results in the following PROPATH:

```
/home/mydir,,/dlc
```

In this case, the second comma indicates to Progress where to insert the working directory.

5.5.4 Compiling P-code on the AS/400

For improved performance, execute r-code stream files instead of p-code. Compile p-code into r-code using a Progress 4GL procedure. The Progress 4GL procedure to create r-code must include the following syntax:

```
COMPILE p-code SAVE.
```

When compiling p-code, Progress/400 lists any compilation errors in the job log with the word "note" and an error number.

Follow these steps to compile code on the AS/400:

1 ♦ Execute your Progress 4GL compile procedure.

2 ♦ Check the job log on the AS/400 for any errors beginning with the word "note."

3 ♦ Fix any Progress code errors on the remote client, check syntax, and repeat the code transfer.

4 ♦ Repeat these steps until you have a clean compile.

See Chapter 6, "Using the Progress/400 Native 4GL Client," or Chapter 7, "Using the Progress/400 AppServer," for specific client details.

5.6 Internationalizing the Native Clients

The native clients use the standard Progress internationalization mechanism. The `convmap.cp` file is shipped with the Progress/400 product. You can FTP the `convmap.cp` files from the Windows or UNIX environment to the AS/400.

5.7 Internationalization Startup Parameters

In order to run the native clients you must be familiar with the defaults for the `-cp*` startup parameters. The defaults in the AS/400 environment are different from the defaults in the Windows and UNIX environments, as Table 5–6 illustrates.

Table 5–6: Defaults for -cp* Startup Parameters *(1 of 2)*

Parameter	Default for Windows/UNIX	Default for AS/400
-cpcase	Basic	BASIC
-cpcoll	Basic	BASIC
-cpinternal	ISO8859-1	IBM037
-cplog	Obtained from -cpinternal	Obtained from -cpinternal
-cpprint	Obtained from -cpstream	Obtained from -cpstream

Table 5–6: Defaults for -cp* Startup Parameters *(2 of 2)*

Parameter	Default for Windows/UNIX	Default for AS/400
-cprcodein	None	None
-cprcodeout	None	None
-cpstream	IBM850	IBM037
-cpterm	Obtained from -cpstream	Obtained from -cpstream

The main `-cp*` parameters are `-cpinternal` and `-cpstream`. Some guidelines on these and the other parameters follow:

- **Do not** change the value of `-cpinternal`. Progress/400 is built on an AS/400 using code page IBM037, and the product uses this internal code page value.

- The `-cpstream` parameter tells Progress the code page of the source code (.p) and the output (to printer or file). Because both the source code and the output involve operating system objects, it is expected that `-cpstream` matches the operating system code page used by the AS/400. If your program uses a different code page, you must specify it when you start up the native clients. See Chapter 6, "Using the Progress/400 Native 4GL Client," for the syntax to start up the Native 4GL Client. See Chapter 7, "Using the Progress/400 AppServer," for instructions on how to start up the AppServer.

  If you do not specify `-cpstream` and the source program contains characters that are not in the IBM037 code page, the compiler might not understand the source code, or some characters might be converted incorrectly in the output.

- As noted, the `-cplog`, `-cpprint`, and `-cpterm` parameters obtain their values from the main `-cp*` parameters.

- The remaining parameters are not relevant in the AS/400 environment.

Observe that the native clients use IBM037 as the default code page value for the `-cpinternal` and `-cpstream` parameters rather than the Progress default code page values of ISO8859-1 and IBM850, respectively.

If you store procedures in the IFS in ASCII format (IBM850) directly from a Windows machine to a mapped drive, you must set `-cpstream` parameter appropriately. In this case, you must set it to either IBM850 or ISO8859-1. For example, the following command writes files into the IFS that can then be read by an ASCII client:

```
OUTPUT TO file CONVERT TARGET "IBM850"
```

For more information about the -cpstream and -cpprint parameters, see the *Progress Startup Command and Parameter Reference*.

6

Using the Progress/400 Native 4GL Client

This chapter documents Progress/400 product information that is specific to the Native 4GL Client. Topics covered in this chapter include:

- Overview of the Native 4GL Client
- Running Progress applications using the Native 4GL Client
- Using QCMD to start the Native 4GL Client remotely
- Passing parameters to the Native 4GL Client
- Executing triggers
- Preserving the working directory
- Calling Progress 4GL programs from OS/400 HLL programs

6.1 Overview

The Native 4GL Client allows you to run Progress applications on the AS/400. Like other Progress/400 utilities on the AS/400, it has a standard OS/400 command interface. You execute the Native 4GL Client as you would any OS/400 command. What makes the Native 4GL Client different from other Progress clients is that it has no interactive capabilities. With this client, applications creating reports or doing large updates to the DB2/400 database can run locally on the AS/400, thus taking advantage of the AS/400 server performance and reducing the load on network resources.

6.2 Running Progress Applications Using the Native 4GL Client

As discussed in Chapter 5, "Preparing to Use AS/400-based Clients," follow these steps to run your Progress applications on the AS/400:

1 ♦ Install Progress/400 on the AS/400 and install Progress on the client machine.

2 ♦ Create a dictionary library that contains a schema image.

3 ♦ Transfer your Progress application to the AS/400.

4 ♦ Invoke the Native 4GL client to execute your Progress 4GL code (p-code or r-code) using the STRPROCLI utility.

To start the Native 4GL Client, execute the STRPROCLI utility and provide the procedure name you want to execute. Progress requires the following minimum syntax:

```
STRPROCLI STPROC(Progress-procedure-name)
```

For more detailed information on the STRPROCLI utility, see Chapter 10, "AS/400 Utilities."

6.3 Using QCMD to Start the Native 4GL Client Remotely

You can execute the Native 4GL Client code from a remote client connected to a DB2/400 database. Within a Progress procedure on a remote client, use the following syntax to execute your code:

```
DO TRANSACTION:
    CREATE QCMD:
    ASSIGN QCMD.cmd = "STRPROCLI SCHDB(DB-NAME) STRPROC(CODE)".
    VALIDATE QCMD.
END.
```

6.4 Passing Parameters to the Native 4GL Client

You might need to pass parameters from the client or the server to p-code running on the Native 4GL Client. The following lists several methods:

- Use QCMD to activate a Native 4GL Client by running the STRPROCLI utility. You can also call a CL program that runs this utility. For a description of this utility, see the "Start Native 4GL Client (STRPROCLI)" section in Chapter 10, "AS/400 Utilities."
- Use one of the following:
  - Data queues. See the "Data Queue Support" section in Chapter 11, "Progress 4GL Interfaces to OS/400 Languages and Objects," for more details.
  - Data areas. See the "Data Area Support" section in Chapter 11, "Progress 4GL Interfaces to OS/400 Languages and Objects," for more details.
  - PROCALL and the EPI interface. See the "External Programming Interface (EPI)" section in Chapter 11, "Progress 4GL Interfaces to OS/400 Languages and Objects," for more details.

6.5 Executing Triggers

The Native 4GL Client executes triggers defined in the Progress/400 Data Dictionary. However, for the triggers to work successfully with the Native 4GL Client, you must transfer the trigger p-code to the AS/400 after migrating the database to the AS/400. When you transfer the p-code, you must make sure it resides in the same directory name as the Windows client. For example, if your trigger code resides in `\current-working-directory` on your remote client, you must have your trigger code reside in a parallel directory on the AS/400.

6.6 Preserving the Working Directory

The Native 4GL Client constructs the Progress/400 PROPATH as Progress does on other platforms. Progress/400 adds the path value (specified in the STRPROCLI utility) to the front of the default PROPATH value `',/(install-directory)'`. The preceding comma (,) in this default PROPATH value indicates that the working directory also will be prepended. It is possible to lose the working directory in the PROPATH if you do not specify enough commas in the PROPATH value.

For example, if you specify `'/home/mydir'` as PROPATH on the STRPROCLI command line, this is prepended to the default PROPATH of `',/(install-directory)'` to produce `'/home/mydir,/(install-directory)'`. The single comma acts as a path delimiter and the working directory is lost. If you do want the working directory in the middle of the PROPATH,

then the PROPATH value on the STRPROCLI command line should be '/home/mydir,' resulting in a PROPATH of '/home/mydir,,(*install-directory*)/'. In this case, the second comma indicates the working directory.

6.7 Calling Progress 4GL Programs from OS/400 HLL Programs

Progress/400 provides a set of commands that allow OS/400 HLL (High Level Language) programs to call Progress 4GL programs running on the Native 4GL Client. This set of commands does the following:

- The PROCALL program calls the Progress 4GL program.
- The EPI ENTRY and EPI EXIT commands allow parameters to be input to the 4GL program and output to the HLL caller.

  For more information on EPI, see Chapter 11, "Progress 4GL Interfaces to OS/400 Languages and Objects."

6.7.1 The PROCALL Program

The PROCALL program allows the OS/400 HLL to call and pass parameters to a Progress 4GL program. PROCALL accepts up to 32 parameters. The first is the name of the called program, and the remaining parameters are used to pass input to and output from the Progress 4GL program.

The following CL and RPG programs demonstrate how to use the PROCALL program. The Progress 4GL program demonstrates how the 4GL handles entry parameters:

CL Program That Calls a Progress Program

```
PGM
DCL     VAR(&DOTP) TYPE(*CHAR) LEN(128)
DCL     VAR(&PARM1) TYPE(*CHAR) LEN(50)
DCL     VAR(&PARM2) TYPE(*DEC) LEN(15 5)
CHGVAR VAR(&DOTP) VALUE('/anydir/sample.p')
CHGVAR VAR(&PARM1) VALUE('OLD VALUE')
CHGVAR VAR(&PARM2) VALUE(12.345678)
CALL    PGM(PROCALL) PARM(&DOTP &PARM1 &PARM2)
ENDPGM
```

RPG IV Program That Calls a Progress Program

```
I   '/anydir/sample.p'     C     DOTP
C*
C             CALL   'PROCALL'
C             PARM   DOTP     PROC  128
C             PARM   'HELLO'   PARM1 50
C             PARM   16.654    PARM2 150
C*
```

Progress Program sample.p

```
DEFINE NEW SHARED VARIABLE sts AS INTEGER.
DEFINE NEW SHARED VARIABLE msgarr AS CHARACTER EXTENT 1.
DEFINE NEW SHARED VARIABLE parm1 AS CHARACTER.
DEFINE NEW SHARED VARIABLE parm2 AS DECIMAL.
OS400 EPI STATUS(sts) MESSAGES(msgarr)
      ENTRY PARM(parm1 AS CHARACTER(50) USE INPUT-OUTPUT)
            PARM(parm2 AS DECIMAL(15, 5) USE INPUT-OUTPUT).
OUTPUT TO PRINTER.
DISPLAY parm1 parm2 WITH DOWN.
DOWN.
parm1 = "NEW VALUE".
parm2 = 87.65432.
DISPLAY parm1 parm2.
OUTPUT CLOSE.
OS400 EPI STATUS(sts) MESSAGES(msgarr) EXIT.
```

PROCALL Parameters

PROCALL can accept up to 32 parameters. The first parameter must be the name of the Progress 4GL program name that you want to call. You must pass a name that conforms to IFS naming conventions. If the stream file does not exist, Progress/400 displays an error and the call to PROCALL does not complete normally. PROCALL is designed primarily for RPG and CL, hence the first parameter must be defined as a 128-byte character field and must be space padded on the right. It cannot be NULL terminated.

Progress/400 considers any other parameters passed to the PROCALL program to be parameters to the 4GL program. These parameters must also be passed using the RPG or CL standards. The following rules apply:

- All parameters passed to a 4GL program must be declared in the calling program exactly as they are declared in the called Progress 4GL program. You use the EPI ENTRY command to declare the parameters input to the 4GL program. See the "EPI ENTRY Command" section for details.

- Character strings passed to the 4GL program must be declared as a fixed length; they cannot be NULL terminated. In the previous examples, &PARM1 is defined as a 50-byte character string in the RPG and CL programs, and it is declared as a 50-byte character string in the EPI ENTRY command in the Progress program.

- Just as in a CL or RPG program, a Progress 4GL program can accept numeric parameters. The numeric types supported are packed, zoned, and integer.

If you pass more than 32 parameters to PROCALL, it returns an error.

For a table of rules to follow when defining the mapping between Progress and OS/400 data types, see the "DB2/400-to-Progress Data Type Mapping" section in Chapter 2, "Common Product Information."

EPI ENTRY Command

When you call a Progress 4GL program, the external parameters passed to the 4GL are defined using the EPI ENTRY command. Progress uses the same syntax for the EPI ENTRY command and for the EPI CALL command, except that with EPI ENTRY you need not specify the PGM parameter.

Once you run the EPI ENTRY command, the shared variables set up in the PARM parameters are assigned the values passed from the PROCALL program. You define parameters for EPI ENTRY using the PARM parameter. The USE parameter of the PARM keyword has the following meaning:

- A USE of INPUT indicates that the parameter is input to the 4GL program.

- A USE of OUTPUT indicates that the parameter is output from the 4GL program to the caller.

- A USE of INPUT-OUTPUT indicates that the parameter is either input to or output from the 4GL program as needed.

Here is an example of the EPI ENTRY command:

```
OS400 EPI STATUS(sts) MESSAGES(msg)
ENTRY
PARM(prog-var[extent] AS TYPE(len, decimals) USE use)
```

EPI EXIT Command

Use the EPI EXIT command to send parameters back to the caller. The command does not have parameters; its only purpose is to save the shared variable values declared in the EPI ENTRY command so that the calling HLL program can retrieve them.

7

Using the Progress/400 AppServer

This chapter documents how to use the Progress/400 AppServer product on the AS/400. For complete documentation on AppServer concepts, configurations, and programming considerations, see the *Progress Installation and Configuration Guide Version 9 for UNIX* and *Building Distributed Applications Using the Progress AppServer* before reading this chapter. This chapter covers the following Progress/400 topics:

- Overview
- Task List
- Managing the Progress/400 AppServer product

7.1 Overview

The Progress/400 AppServer product is comprised of three components: the Progress/400 AdminServer, the Progress/400 NameServer, and the Progress/400 AppServer instance. The Progress/400 AdminServer manages the Progress/400 NameServer, the Progress/400 AppServer instances, and licensing. The Progress/400 NameServer mediates client connections for a Progress/400 AppServer instance. Each Progress/400 AppServer instance runs the 4GL on the AS/400 for a remote AppServer client. A Progress/400 AppServer instance cannot talk to any other AppServer instance. The following sections explain the framework for these three components on the AS/400.

Progress Software Corporation recommends using the Progress Explorer tool to manage the Progress/400 AppServer product.

7.1.1 The Progress/400 AdminServer

The Progress/400 AdminServer is made up of two AS/400 jobs. The first job, named ADMINSERV, controls the Java Virtual Machine. The second job is the Java Virtual machine itself, named QJVACMDSRV. To start the AdminServer, use the Progress/400 STRADMSVR command. This command starts the AdminServer subsystem using the configured subsystem name and also starts the AdminServer jobs. For more information on the STRADMSVR command, see Chapter 10, “AS/400 Utilities.”

The AdminServer jobs on the AS/400 run in the subsystem defined in the Administration Subsystem field within the Progress/400 AppServer Defaults menu of the Progress/400 install program. This subsystem can also be defined with the Change Progress/400 Defaults command (CHGPRODFT). For more information on the CHGPRODFT command, see Chapter 10, “AS/400 Utilities.”

Figure 7–1 diagrams the Progress/400 AdminServer with its required jobs.

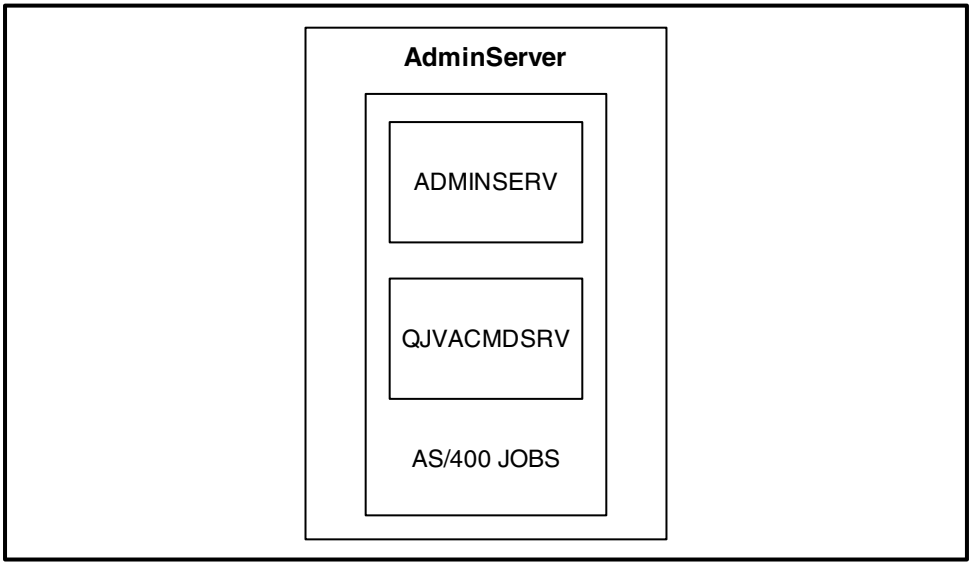


Figure 7–1: Progress/400 AdminServer Architecture

The Progress/400 AdminServer shown in Figure 7–1 runs within the Administration Server subsystem specified at install time.

7.1.2 The Progress/400 AppServer Instance

The Progress/400 AppServer instance contains the same components as the AppServer instance on other Progress platforms. It includes both the Application Broker and the Application Server. On the AS/400, these components run as OS/400 jobs.

Every Application Broker on the AS/400 is made up of two OS/400 jobs. The Application Broker jobs follow a specific naming convention. Progress/400 gives the first job the same name as the name of the Application Broker, for example ASBROKER1, and it controls the Java Virtual Machine. The second job, the actual Java Virtual machine, always takes the name QJVACMDSRV. These two jobs together make up one Application Broker.

Every Application Broker can start a predefined number of Application Servers. Each Application Server on the AS/400 is made up of two AS/400 jobs that also follow a specific naming convention. The first job, named QZSHSH, binds together the Application Broker and the Application Server that the broker controls. The second job, named QZP0SPWT, is the actual server job. This job executes the Progress/400 AppServer program called PROAPPSVR.

If any of these jobs within the Application Broker or the Application Server ends, its corresponding partner job will end as well.

Figure 7–2 diagrams the Application Broker and a single Application Server with their required jobs.

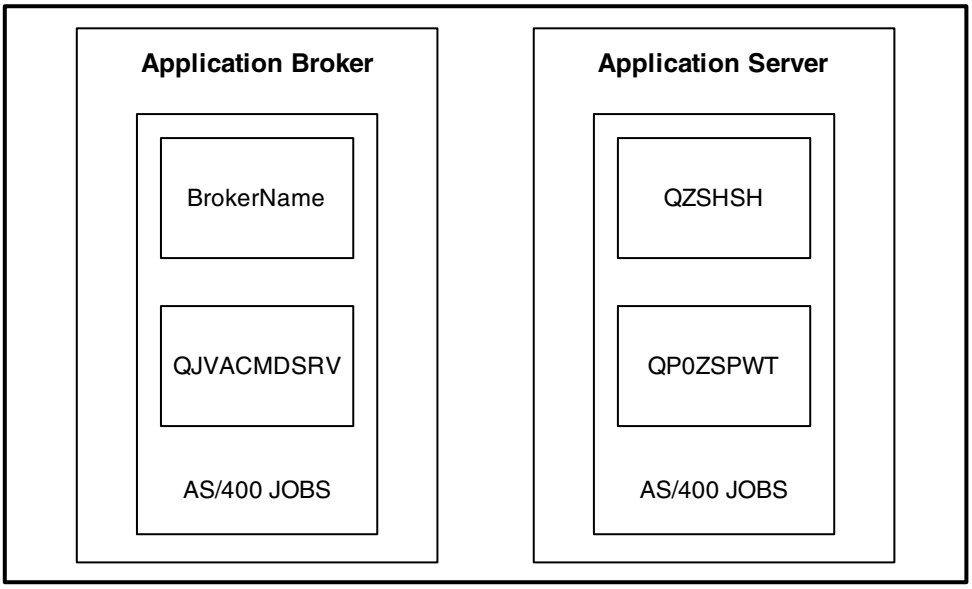


Figure 7–2: Progress/400 AppServer Instance Architecture

As shown in Figure 7–2, the Application Broker and Application Server jobs make up one Progress/400 AppServer instance. A Progress/400 AppServer instance runs in the subsystem specified at install time.

7.1.3 The Progress/400 NameServer

As with the Progress/400 AdminServer and Progress/400 AppServer instance, the Progress/400 NameServer also contains two AS/400 jobs, which follow specific naming conventions. The first job that starts uses the same name as the Progress/400 NameServer, for example NS1. This job controls the Java Virtual machine. The second job that starts, named QJVACMDSRV, is the Java Virtual machine itself. The two Progress/400 NameServer jobs always run in the Progress/400 Administration Server subsystem, specified during the Progress/400 AppServer product installation.

If configured to auto start in the `ubroker.properties` file, the Progress/400 NameServer starts when the AdminServer starts. You can also configure and start it remotely using the Progress Explorer on NT or using the NSMAN utility on UNIX or NT. See the "Managing the Progress/400 AppServer Product" section in this chapter for more details.

Figure 7–3 diagrams the Progress/400 NameServer with its required jobs.

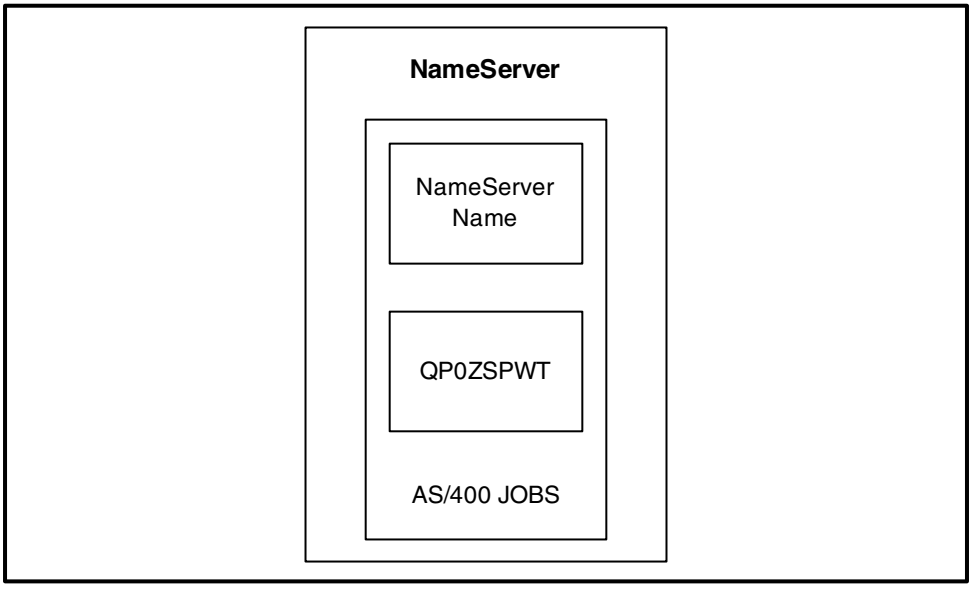


Figure 7–3: NameServer Architecture

The Progress/400 NameServer shown in Figure 7–3 always runs in the Progress/400 Administration Server subsystem.

7.2 Task List

Follow these steps on the AS/400 to use the Progress/400 AppServer product:

1 ♦ Install the Progress/400 AppServer product on your machine.

2 ♦ Optionally, modify the NameServer and Progress/400 AppServer instance configurations in the `ubroker.properties` located in $DLC/properties using EDTF.

3 ♦ Add the Progress library to your library list.

4 ♦ Start the AdminServer using the Progress/400 command STRADMSVR. Any Auto-start servers (Name or AppServer) will also start at this time.

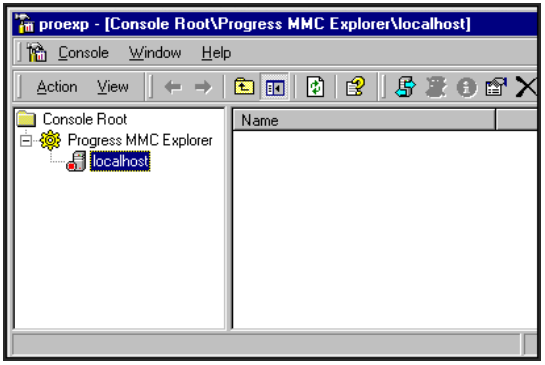
Use the WRKACTJOB command to make sure the ADMINSERV starts under the user you specified. Look at other jobs in the subsystem and wait for their status to stabilize. New jobs will appear and disappear for a few minutes.

Follow these steps on an NT client machine once you have started the AdminServer on the AS/400:

1 ♦ Ensure that you have the Progress Explorer installed and running on your NT machine.

2 ♦ Choose start→ Programs→ Progress→ Progress Explorer from your Windows Taskbar.

The Progress Explorer main window appears. When you expand the tree view under Progress MMC Explorer, you see a list of machines where you can start a Progress server. In this example, only the local host machine is listed:



To add your remote AS/400 to the list, do the following:

a) Select Progress MMC Explorer from the tree view.

b) Choose Action→ Add Progress Server from the main menu.

c) Enter the server name (the name of a remote machine), your user name, and your password in the Server Properties dialog box.

d) Select Save username and password if you want to automatically log into this machine every time you connect to it.

e) Select the Advanced tab and make sure that the Server Port Number matches the Default AdminServer Port Number specified during the Installation or when using the CHGPRODFT command.

f) Select Reconnect at startup only if you want to automatically connect to this machine every time you start Progress Explorer.

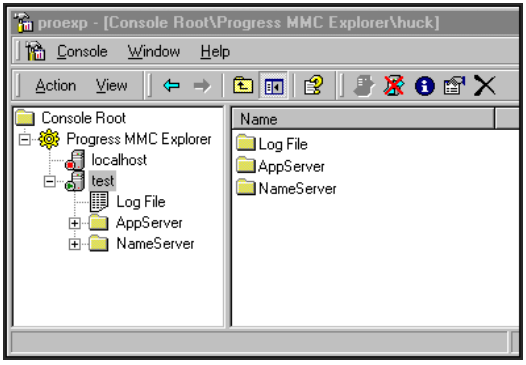
g) Choose OK.

3 ♦ Select the name of the machine where your AdminServer is running.

You will see a login dialog box, unless you configured the Progress Explorer tool to automatically log in to a particular machine.

4 ♦ Choose Action→ Connect from the main menu.

5 ♦ Expand the tree view of the connected server machine. The expanded tree view will look similar to the following:



You can now manage the Progress/400 AppServer as you would any other AppServer. You can end all AppServers and Name Servers from your client using the Progress Explorer tool or using the respective command-line utilities ASBMAN and NSMAN.

You can shut down an AdminServer process at any time from the AS/400. If you do shut down an AdminServer process, all other jobs managed by this AdminServer end as well. To shut down the AdminServer, use the Progress/400 ENDADMSVR command with the appropriate user profile and password.

7.3 Managing the Progress/400 AppServer Product

You must start the Progress/400 AdminServer locally, using the Start AdminServer command (STRADMSVR), but you must manage the Progress/400 AppServer product remotely. You can use the Progress Explorer tool on NT to manage the Progress/400 AppServer product once the AdminServer has started, or you can use the appropriate management utilities from either UNIX or NT. For an explanation of how to manage an AppServer using the Progress Explorer tool or using the remote UNIX or NT management utilities (ASBMAN, ASCONFIG, NSMAN, NSCONFIG), see the *Progress Installation and Configuration Guide Version 9 for UNIX*. For AS/400-specific management information, read this chapter.

7.3.1 Starting the Progress/400 AdminServer

Start the Progress/400 AdminServer with the local STRADMSVR command using the following syntax on the AS/400:

```
STRADMSVR
```

7.3.2 Modifying the Configurations

Progress/400 stores the configurations for all the components managed by the Progress/400 AdminServer, including the Progress/400 NameServer, and the Progress/400 AppServer instances in the `ubroker.properties` file. The Progress/400 `ubroker.properties` file follows the same format and standards as the `ubroker.properties` file on UNIX (for example, directory path separators and environment variables). For a detailed description of this file, its function, customization, and verification, see the *Progress Installation and Configuration Guide Version 9 for UNIX*.

As with UNIX, the Progress/400 `ubroker.properties` file resides in the $DLC/properties directory. On the AS/400, this directory is located within the root file system. For more information on the root file system, see Chapter 5, "Preparing to Use AS/400-based Clients." Follow the same guidelines as UNIX for managing this file. To edit the `ubroker.properties` file with the specific configurations for your Progress/400 AdminServer, Progress/400 NameServer, and Progress/400 AppServer instance, choose from the following options. Always make a copy of the `ubroker.properties` file, edit the copy, and verify the results before replacing the original with your edited copy:

- **EDTF (Edit File)** — Edit the file on the AS/400 using the command EDTF. The EDTF command ships with the OS/400 Version V4R4. Progress/400 also provides this command for use on earlier versions of the OS/400.

  NOTE: If you use EDTF to edit your file, you must also verify your changes with the configuration validation utilities on either NT or UNIX. If your AdminServer is running and you choose the Progress Explorer to verify your changes, you no longer can use EDTF, as the Progress Explorer makes changes to the end-of-line (EOL) characters.

- **Progress Explorer tool** — If your AdminServer is running, you can edit the file on the AS/400 from an NT interface using the Progress Explorer. The Progress Explorer also verifies any changes you make to the Progress/400 `ubroker.properties` file. If you choose this method, you can no longer use EDTF, as the Progress Explorer changes EOL characters. This is the recommended editing method.

- **UNIX editor** — Edit the file on UNIX, verify your changes with the UNIX configuration validation utilities, and then move the file back to the AS/400. You can use this method if the AdminServer is not running.

- **Windows editor** — Edit the file on NT with Client Access. This requires that you map a drive to the root directory on the AS/400. Verify your changes with the NT configuration validation utilities, and then move the file back to the AS/400. You can use this method if the AdminServer is not running.

7.3.3 Starting the Progress/400 NameServer

Start the Progress/400 NameServer using the Progress Explorer or the remote management utility NSMAN. If you have configured the Progress/400 NameServer to auto-start in the Progress/400 `ubroker.properties` file, you can start it using the STRADMSVR command.

7.3.4 Starting the Progress/400 AppServer

Start the Progress/400 AppServer using the Progress Explorer or the remote management utility ABSMAN. Or, if you have configured the Progress/400 AppServer to auto-start in the Progress/400 `ubroker.properties` file, you can start it using the STRADMSVR command.

7.3.5 Managing the Progress/400 NameServer and Progress/400 AppServer

Progress/400 does not provide local utilities for managing the Progress/400 NameServer or Progress/400 AppServer instances. Therefore, you use either the Progress Explorer tool or remote management utilities to query the status of any running Progress/400 NameServer or Progress/400 AppServer instances. You can also use the Progress Explorer tool to shut down any running Progress/400 NameServer or Progress/400 AppServer instance.

7.3.6 Stopping the Progress/400 AdminServer

You can stop the Progress/400 AdminServer using the ENDADMSVR command on the AS/400 with the appropriate user profile and password.

To stop the AdminServer with the ENDADMSVR command, use the following syntax on the AS/400:

```
ENDADMSVR user(username) password(password)
```

7.3.7 Progress/400 AppServer Instance Job Information

All of the AS/400 jobs for the Progress/400 Administration Server, the Progress/400 Name Server, and the Progress/400 AppServer instance start up using the user's name that was specified in the Progress/400 AppServer Defaultsconfiguration menu at install time (for example, PROGRESS). This user must have *ALLOBJ authority for the Progress/400 AppServer product to perform correctly.

The following values specify the environment where the Progress/400 AppServer instance will run. You set these values at install time, or with the CHGPRODFT command. For more information on the CHGPRODFT command, see Chapter 10, "AS/400 Utilities."

*ADMSVR

If you select *ADMSVR in the AppServer Instance Environment field, each Progress/400 AppServer instance will start its jobs in the Progress/400 AdminServer subsystem. For example, if you specify APPSVR in the Administration Jobs Subsystem Name field, then when an Application broker starts, its jobs (for example, ASBROKER1 and QJVACMDSRV) will start in the APPSVR subsystem. Any Progress/400 Application Server jobs started by this Application Broker will also start in the APPSVR subsystem.

*BROKER

If you select *BROKER in the AppServer Instance Environment field, each Progress/400 AppServer instance will start its jobs in its own subsystem with the same name as the Application Broker. For example, if the Application Broker, ASBROKER1, starts and the install subsystem is *BROKER, Progress/400 creates a subsystem with the name ASBROKER1, and the broker jobs ASBROKER1 and QJVACMDSRV start within this subsystem. Any Application Server jobs for ASBROKER1 will also start in this same subsystem. Any additional Application Brokers started will have their own subsystem and their jobs will run within the subsystem created with their name. If Progress/400 finds an existing subsystem of the same name as the broker, it will use this existing subsystem. Otherwise, Progress/400 creates a new subsystem using the broker name as the subsystem.

*USER

If you select *USER in the AppServer Instance Environment field, each AppServer instance will start its jobs in an existing subsystem specified by the job queue and job description parameters.

7.3.8 Progress/400 AppServer Instance Environment

Depending on what you select in the Progress/400 AppServer Defaults field during installation or using the CHGPRODFT utility, you can allow for complete Progress/400 control over the jobs and resources or complete user control. There are certain factors to consider when deciding the method to use for your particular application.

It is important to consider the size of the AS/400 machine before choosing a subsystem method for your application's use. The more subsystems your application starts, the more memory your machine will need. The different subsystem methods will also affect performance.

The choice of the *ADMSVR method gives Progress/400 complete control over where the jobs run. With this selection, all of the jobs run in the one subsystem specified by the user at install time. Progress/400 manages this subsystem and all jobs started within it.

The choice of the *BROKER method causes Progress/400 to use more resources as it creates a subsystem for each AppServer instance started. This method allows the user to tune each subsystem to individual specifications.

The choice of the *USER method gives the user complete control over where the jobs run and what resources are allocated to the subsystem specified.

7.3.9 Naming Multiple Servers

Progress/400 supports running multiple AdminServers, NameServer, and AppServer instances on a single machine. To do this you must make each unique, as the "Naming Conventions" section in this chapter documents.

AdminServer Port Number

The Progress/400 AdminServer has a listening port for remote clients. Progress/400 assigns a default value of 20931 to this port. If you choose to have multiple Progress/400 AdminServers, you must give each a unique port number to using the Change Progress/400 Defaults utility (CHGPRODFT). For more information on the CHGPRODFT command, see Chapter 10, "AS/400 Utilities."

NameServer Name and Port Number

The Progress/400 NameServer has a name and a listening port for remote clients. Progress/400 assigns this NameServer a default name of NS1 and gives it a default port value of 5162. If you choose to have multiple Progress/400 NameServers, you must give each a unique name and port number by editing your `ubroker.properties` file.

AppServer Broker Name and Port Number

The Progress/400 AppServer Broker has a name and a listening port for remote clients. Progress/400 assigns this broker a default name of ASBROKER1 and gives it a default port value of 3050. If you choose to have multiple Progress/400 AppServer instances, you must give the AppServer Broker for each instance a unique name and port by editing your `ubroker.properties` file.

AppServer Instance Port Number

The Progress/400 AppServer has a listening port for remote clients. Progress/400 assigns a default port value of 2000 to the first Progress/400 AppServer instance. If another AppServer instance starts, Progress/400 automatically creates a unique and sequential port value. You can change the port value for the Progress/400 AppServer instance by editing your `ubroker.properties` file.

7.3.10 Naming Conventions

The Application Broker and Progress/400 NameServer names should be no more than 10 characters and be unique. If you use a name with more than 10 characters, Progress/400 truncates the name to 10 characters. This could lead to unresolved names if your application specifies a larger name than Progress/400 uses. Because Progress/400 uses these names for subsystem and job name lookups, the names must also be unique. If you do not make these names unique, two jobs with the same name could run simultaneously at the same time leading to confusion.

8

System Administration

This chapter discusses the following:

- Working with Progress jobs
- Database security
- Transaction control
- Code pages and collation support
- DataServer performance
- Progress/400 settings file
- Progress/400 attributes
- Test and production environments
- Reserved Words

8.1 Working with Progress Jobs

Progress/400 provides job control for the jobs it starts and manages through the Work With Progress Jobs (WRKPROJOB) utility. Using this utility, you can display information about, and exercise limited control over, all Progress/400 jobs.

When Progress/400 broker, server, or native clients' processes start, user spaces are created in the Progress Temporary Library (PROTMP). The WRKPROJOB utility uses these user spaces to determine what Progress jobs are currently active. Normally, Progress deletes these temporary objects at the end of the job that created them. However, if the AS/400 shuts down abnormally, these temporary objects are not deleted from the PROTMP library, and WRKPROJOB shows that the jobs that created them still exist, though the job status (Job STS) field for such a job is blank. To remedy this problem, after the AS/400 reboots (completes its Initial Program Load, or IPL) after the abnormal shutdown, you must delete all of the "status" user spaces in the PROTMP library. To do this, follow these steps:

1 ♦ Enter the following command:

```
WRKOBJPDM LIB(PROTMP) OBJ(PRO*) TYPE(*USRSPC)
```

2 ♦ Type **4** next to each user space in the object list that is displayed.

For a complete description of WRKPROJOB, see the "Work with Progress Jobs (WRKPROJOB)" section in Chapter 10, "AS/400 Utilities."

8.2 Security

The following sections discuss Progress/400 Database and AppServer security on the AS/400.

8.2.1 Database Security

In standard Progress, the Data Dictionary handles both database and application security. When you use Progress against a DB2/400 database, the DB2/400 security facilities handle database security, but your applications must handle application security. Before reading this section, you should understand OS/400 user profiles and OS/400 object authority. For more information about OS/400 security, see the *AS/400 Security Concepts and Planning Guide*.

Implementing Progress/400 Security

The AS/400 handles user security through a user profile that identifies each user to the system. The Progress/400 DataServer implements user and object security by using OS/400 user profiles to start an individual database server process (or job) on the AS/400.

When a Progress user makes a connection to a DB2/400 database, the Progress client passes the necessary user-profile information to OS/400.

For remote clients, you must provide the User ID (-U) and Password (-P) parameters at connection time:

- The -U and -P parameters correspond to an OS/400 user profile and password, not a Progress user ID and password.
- For host-based (5250 nonprogrammable workstation) users, Progress uses the user profile assigned to the active 5250 session.

After the client passes user-profile information and attempts to connect, the following occurs:

1. The AS/400 verifies that the Progress user's OS/400 user profile is valid.

   If the user profile is not valid, the client cannot connect and receives an error message stating that the server rejected the login attempt.

2. If the user profile is valid and has appropriate program-object authorities to the evoke program for the Progress/400 DataServer programs as specified in the program start request, the AS/400 verifies the user's object authority for the database object being accessed:

   - If the database-object authority is valid for the operation that the client wants to perform, the database server performs the operation.
   - If the authority is invalid, the client is denied permission to perform the attempted operation and Progress generates an error message.

In addition to the OS/400 security, you might also want to consider the following techniques to ensure security:

- Application security
- Procedures for setting up a user permissions file on the AS/400

For more information, see the "Application Security" section in Chapter 2, "Common Product Information."

8.2.2 AppServer Security

For general AppServer security guidelines, see *Building Distributed Applications Using the Progress AppServer*.

8.3 Transaction Control

Transaction control, known as commitment control on the AS/400, is a mechanism that allows you to undo transactions and restore a database to its prior state. Commitment control protects you from writing incomplete changes to a database. Progress databases use a before-image file (.bi) to handle transaction control automatically, but Progress/400 does not.

The OS/400 handles commitment control automatically when you start journaling on a file. When an application opens database files with commitment control on, transactions can be applied or removed as transaction units. Each DB2/400 database file must have an active journal receiver for commitment control to be in effect. The *journal receiver* (*JRNRCV) is an OS/400 object that contains information about changes made to a DB2/400 physical file. The records of these changes are called *journal entries*. For information on starting and maintaining journaling, see the "Journaling" section.

The Progress/400 DataServer does not manage journal receivers or the journaling process, and it does not support Progress 2-phase commit.

8.3.1 Local Before-imaging

The DataServer allows you to work with either commitment control on the AS/400 server or local before-imaging on the remote Progress client. Local before-imaging should not replace commitment control on the AS/400. It provides single-user UNDO processing; it does not provide multi-user transaction control. Use it only if commitment control is **not** in effect for DB2/400 database files. If the client were to fail, you would not be able to undo transactions or recover your database.

NOTE: Commitment control on the AS/400 server and local before-imaging on the client are incompatible. Use only one mechanism.

If you do not use journaling on your DB2/400 files, you can use the local before-image file to back out transactions if the remote client continues to run. Progress Software Corporation recommends that you use DB2/400 journaling and commitment control.

By default, the Progress/400 DataServer starts OS/400 commitment control but will open DB2/400 files even if commitment control is not in effect for them. See the "Specifying Transaction Control Techniques" section for instructions on specifying a different response to commitment control.

If the file is opened with commitment control on, the DataServer ends commitment control when the Progress session ends. Opening files with commitment control on assures that standard Progress transaction scoping rules apply. (See the "Transactions" section in Chapter 2, "Common Product Information," for information about how the standard Progress RDBMS and OS/400 might view some transactions differently.)

When you use Progress to write to DB2/400 database files that are not journaled, the DataServer writes a message to the OS/400 job log when it opens the nonjournaled file. This is the default behavior for the DataServer, which you can control with the DataServer (-Dsrv TRANSCTL) startup parameter as discussed in the next section.

8.3.2 Specifying Transaction Control Techniques

To use server-based transaction control, start journaling and start the OS/400 commitment control facility.

You can specify remote client-based transaction control with the TRANSCTL option of the DataServer (-Dsrv) connection parameter. This is the syntax for -Dsrv TRANSCTL:

SYNTAX

```
-Dsrv TRANSCTL=value
```

In remote client-based transaction control, the remote client uses a Progress local before-image file (LBI). Table 8–1 describes how to use -Dsrv TRANSCTL to implement transaction control.

Table 8–1: DataServer (-Dsrv) Arguments *(1 of 2)*

Keyword	Description
COMMIT	Specifies that commitment control extends to all of the files in the Progress/400 database. A journal receiver must exist for the file to enable transaction management. If you are not journaling a DB2/400 file, the application does not open the file and returns an error.
LBI	Specifies that the remote clients use a local before-image file to manage main transactions and nested subtransactions. If you use local before-imaging, you cannot use OS/400 journaling; the two are not interchangeable. LBI works for single-user mode only.

Table 8–1: DataServer (-Dsrv) Arguments *(2 of 2)*

Keyword	Description
NONE	Specifies that transaction control is not in effect.
OPTIONAL	Specifies that the DataServer opens a file regardless of whether commitment control is on or off. If commitment control is off for a file, you cannot undo transactions that affect that file. This is the default behavior for the Progress/400 DataServer.

For example, the following CONNECT statement makes sure that a DataServer application does not open any file for which commitment control is not active:

```
CONNECT as4sh -db mydb2400 -H serv1 -N TCP -S sparesrv
  -Dsrv TRANSCTL=COMMIT.
```

8.3.3 Recovery

The same mechanisms that support transaction control assist in recovering a database after a system or medium failure. The Progress database handles crash recovery by using the before-image (BI) file, and additionally, if you specify it, by using the after-image (AI) file.

DB2/400 recovery relies on the *journaling* mechanism to support commitment control. Journaling is not automatic. You must manually start, maintain, and stop it. When journaling is started, DB2/400 writes database I/O and changes, or journal entries, to a DB2/400 object called a *journal receiver* (*JRNRCV). *Journal entries* are records that DB2/400 writes when database access occurs. Journal entries can be before images or after images. When you start journaling, roll-forward recovery (after images) is the OS/400 default. You can provide crash recovery to a database only when each of the database's physical files is journaled.

8.3.4 Journaling

Journaling, the logging of database I/O and changes into a journal receiver to provide crash recovery, is an AS/400 concept. Before you start journaling, you create a journal receiver (*JRNRCV), then a journal (*JRN). You then start journaling explicitly with the OS/400 STRJRNPF CL command for each physical database file that you want to recover. Once you start journaling for a physical file, the system writes journal entries until you explicitly end journaling with the OS/400 ENDJRNPF CL command.

Table 8–2 lists CL commands that are useful for journaling.

Table 8–2: CL Commands for Journaling

Command	Description
CRTJRNRCV	Creates a journal receiver
CRTJRN	Creates a journal
DLTJRN	Deletes a journal
DLTJRNRCV	Deletes a journal receiver
DSPJRN	Displays a journal
ENDJRNPF	Stops journaling a physical file
STRJRNPF	Starts journaling a physical file
WRKJRNA	Works with journal attributes

The following sections explain how to implement AS/400 journaling. For a basic description of AS/400 journaling, see your OS/400 documentation.

Creating a Journal Receiver

Follow these steps to create a journal receiver and start journaling on the AS/400:

1 ♦ Create the journal receiver object. This is the CRTJRNRCV syntax:

SYNTAX

```
CRTJRNRCV JRNRCV(library/journal-receiver-name)
```

2 ♦ Create the journal object. This is the CRTJRN syntax:

SYNTAX

```
CRTJRN JRN(library/journal-name) JRNRCV(library/journal-receiver-name)
```

3 ♦ Now that the journal and journal receiver objects exist, you can start journaling each physical file in your database. This is the STRJRNPF syntax:

SYNTAX

```
STRJRNPF FILE(library/physical-file) JRN(library/journal-name)
```

Repeat this command for each physical file in your database.

Maintaining a Journal Receiver

Once journaling begins, it continues until you explicitly end it. This is an important consideration because journals use system resources.

This is the syntax for the CL command that stops journaling physical files:

SYNTAX

```
ENDJRNPF FILE(library/physical-file) JRN(library/journal-name)
```

You might want to set up a regular backup schedule for your journals. As journal receivers fill up, you might want to back them up to tape or other media, then delete them from the system. See the *AS/400 Backup and Recovery Guide* for more information about creating a system maintenance and backup schedule for journal receivers.

8.4 Code Pages and Collation Support

The AS/400 provides different character encoding and database record sorting functionality than on most other Progress environments:

- The AS/400 encodes characters using EBCDIC, while most other Progress environments use ASCII character encoding.
- On the AS/400, the sorting of database records is controlled primarily by alternate collating sequence tables, while in other Progress environments, it is controlled by the CONVMAP collation table.

The Progress/400 product provides special code page and data translation support that allows you to make database record sorting on a Progress/400 database operate in the same way as on a Progress database.

If you have experienced sorting problems when running your application, this section provides information that is important to you:

- A FOR EACH or OPEN QUERY statement retrieves records in a different order when executed on the AS/400 server than on the Windows or UNIX client.
- The AS/400 considers lowercase and uppercase letters to be different letters.
- The AS/400 considers the á and the a to be different letters.

NOTE: The discussion in this section assumes that your client is running on Windows; however, the information generally is the same for a client running on UNIX.

8.4.1 Code Page Overview

Before you use the Progress/400 code page and collation functionality, make sure that you understand the difference between code pages and collation tables:

- A *code page* contains the numeric codes assigned to the various characters when they are stored in databases.
- *Collation* specifies the order in which characters are sorted for indexes, queries, and so forth; that is, the *collation sequence*. A *collation table* contains this sort-order information. In the AS/400 environment, collation tables are called alternate collating sequence tables, represented by the ALTSEQ keyword in the DDS specification for database files.

If you have written Progress code based on the ASCII collation sequence, you might need to modify it when you move it to the AS/400. For example, in EBCDIC collation, letters are sorted before numbers, while in ASCII, numbers are sorted before letters. Also, the encoding of the letters is not consecutive in EBCDIC, as it is in ASCII. A common technique when using ASCII collation is to code a range from 'a' to 'z' in order to include all letters. This technique might not function as expected with EBCDIC, because EBCDIC includes additional records in the specified range. The remote client can do EBCDIC collation by using the convmap file. The EBCDIC collation table must be added to the `convmap.cp` file. For information, see the *Progress Internationalization Guide*.

Progress/400 resolves these differences by converting automatically between code pages. To enable code-page conversion, do the following:

1. Identify the code pages needed, based on your application requirements.
2. If necessary, create an alternate collating sequence table.
3. Assign the alternate collating sequence (ALTSEQ) tables.

This ensures that all characters look the same everywhere-database, screen, printer-and that all characters are sorted in the correct order. You must do this task before any data is entered into the database. If you make changes after data is entered, the data might become corrupted, and correcting the resulting problem could be very labor intensive.

Identifying Code Pages

To ensure compatibility of operation between environments, you might need to manage as many as nine code pages. These code pages can reside either on the AS/400 server machine or on the client machine, and can include code pages for the client and server operating systems, Progress products, and data files.

The example in Figure 8–1 illustrates the relationships among the code pages. It is based on an environment in which the AS/400 is configured for Spanish using code page IBM284.

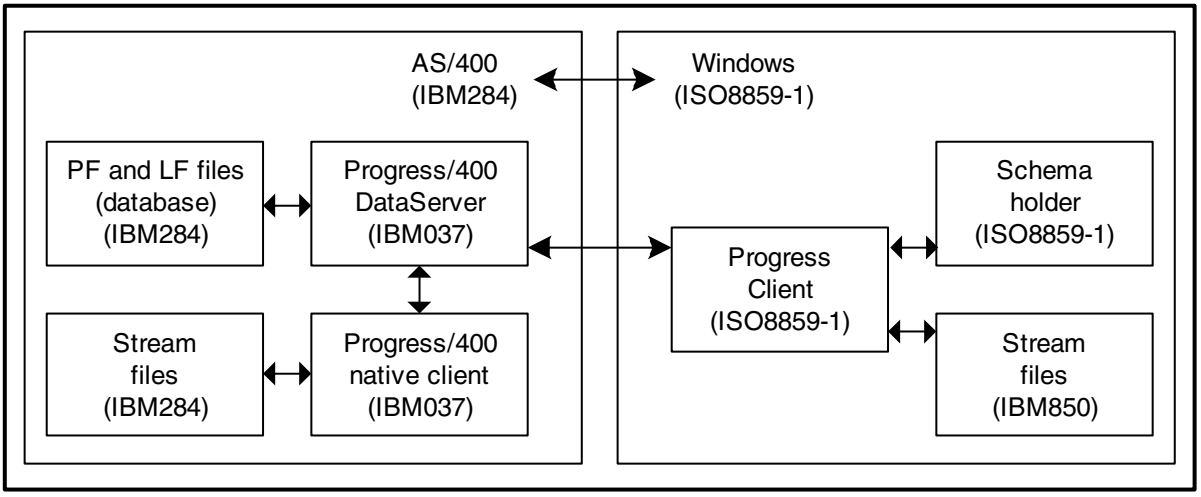


Figure 8–1: Server and Client Code Page Relationships

The following code pages reside on the AS/400 server machine:

- **AS/400 operating-system code page** — This code page is defined when the operating system is installed on the server machine. To determine which code page the AS/400 is using, execute the following command:

```
DSPSYSVAL QCHRID.
```

- **Progress/400 DataServer internal code page** — Progress/400 uses IBM037 as its internal code page. Do not change this code page; Progress/400 was built on an AS/400 using this code page and requires it.

- **Progress/400 native clients' internal code page** — Progress/400 uses IBM037 as its internal code page. Do not change this code page; Progress/400 was built on an AS/400 using this code page and requires it.

- **Progress/400 database code page (PF and LF files)** — This code page is defined when the *PROEMPTY database (server schema) is created on the AS/400 with the DUPPRODB utility. By default it uses *SYSVAL, which specifies the code page used by the AS/400. To determine which code page a Progress/400 database is using, execute the following commands from the Procedure Editor with the schema holder and AS/400 database connected:

```
FIND FIRST p_db NO-LOCK.
DISPLAY _db_xl-name.
```

- **AS/400 stream file code page** — When the Progress/400 native clients create a stream file, the data that it contains is written to the stream file in the code page specified by the -cpstream startup parameter.

  For more information on AS/400 stream files, see IBM's *Integrated File System Introduction*.

The following code pages reside on the Windows client machine:

- **Windows operating-system code page** — This is the default Windows code page (normally 1252, which is very similar to ISO8859-1). Note that the default code page for an MS-DOS session is IBM850, so you get different results when you edit a file depending on whether you use the MS-DOS EDIT command or the Windows Notepad editor. This is important only if you are importing or exporting data from or to flat operating system files (for example, dumping data contents):

  – If you want Windows format, use the startup parameter -cpstream ISO8859-1.

  – If you want MS-DOS format, use -cpstream IBM850.

- **Progress client internal code page** — This is the code page used internally by Progress clients on Windows; the default is ISO8859-1. You can change this using the -cpinternal startup parameter. To determine which code page a Progress session is using, execute the following command from the Procedure editor:

```
DISPLAY session:cpinternal.
```

- **Schema holder code page** — Since a schema holder is created from an empty database, this code page is the same as for the standard Progress empty database (ISO8859-1). To determine which code page a schema holder is using, execute the following commands from the Procedure Editor with the schema holder connected:

```
FIND FIRST _db WHERE _db-type="progress" NO-LOCK.
DISPLAY _db_xl-name.
```

- **Windows stream file code page** — For information, see the *Progress Internationalization Guide*.

Given the number of code pages required, data can be converted between code pages as many as four times before it arrives at its destination. In the Figure 8–1 example, the following conversions occur:

1. The data is loaded as the IBM850 code page.
2. It is converted to an ISO8859-1 Progress internal code page.
3. It is stored in the AS/400 file as an IBM284 code page.

Alternate Collating Sequence (ALTSEQ) Tables

Alternate collating sequence (ALTSEQ) tables are the AS/400 equivalent of Progress collation tables. ALTSEQ tables define the order used to store indexes and specify how to do data sorting in general. They are important not only because they make it possible to accommodate language differences, but because they handle the two main differences between Progress databases and DB2/400 databases: case sensitivity and EBCDIC/ASCII collation. A common test to confirm whether collation is properly set up is to use a Progress 4GL query to retrieve the same set of records from a Progress database and a DB2/400 database; the test should return the same result for both databases. This is especially important when working with languages other than English, specifically those that include special characters or letters (for example, á).

You define collation differently for Progress clients than you do on the AS/400:

- For remote Progress clients, you define collation in the -cpcoll startup parameter, which points to the appropriate table in the CONVMAP.DAT file. For details, see the description of the -cpcoll parameter in the *Progress Startup Command and Parameter Reference*.
- On the AS/400, you define collation by specifying the ALTSEQ keyword when creating the physical file from the DDS file.

When you create the *PROEMPTY server schema with the DUPPRODB command, specify the ALTSEQ tables so that Progress/400 knows which tables to use when creating the AS/400 database files through the Progress/400 Data Dictionary.

NOTE: Using the wrong ALTSEQ table results in incorrect data sorting.

8.4.2 Setting Up Collation for Progress/400 Databases

After you determine which code pages and ALTSEQ tables you need for your Progress/400 database, you can set up your database files. This section describes how to do this in the following circumstances:

- When working with existing database files that are to be included in the Progress/400 server schema
- When creating new database files with the DUPPRODB utility

Working with Existing Database Files

Read this section if your implementation includes existing physical and logical files that were created using AS/400 tools such as DDS and STRSQL.

Progress/400 requires that all files in a database use the same code page and ALTSEQ tables. If the existing files were defined using an ALTSEQ table, you must specify this table as the value of the ALTSEQ parameter in the DUPPRODB utility. However, if they were defined without an ALTSEQ table specification, you can use any ALTSEQ table in DUPPRODB.

It is possible to convert existing files from one ALTSEQ table to another.

NOTE: You should convert existing files **only** if doing so will not interfere with existing non-Progress applications (for example, RPG or COBOL applications) on the AS/400.

Follow these steps to perform the conversion:

1 ♦ Create a new file using the same record format and the desired ALTSEQ table.

2 ♦ Use the CPYF command to copy the records from the old file to the new file. The command handles the collation conversion. For details on the CPYF command, see your IBM documentation.

3 ♦ Delete the old file.

4 ♦ Rename the new file with the name of the old file.

Note that if you cannot convert all of the physical files, you can convert just the logical files. This avoids the need to copy all of the records, but it leaves the physical files with the original definition relative to the ALTSEQ table. To resolve this problem, simply use the CHGPRODCT utility to notify Progress/400 about the logical files.

Follow these steps to convert a logical file:

1 ♦ Delete the logical file.

2 ♦ Re-create the logical file, changing the definition to the appropriate ALTSEQ table.

Creating a New Progress/400 Database (DUPPRODB)

Read this section if you are creating new database files. It describes how to specify ALTSEQ tables when using the DUPPRODB utility to create a new Progress/400 database.

The example in Figure 8–2 illustrates using the DUPPRODB utility on an AS/400 machine whose system code page is IBM037.

```
 Duplicate Dict & DB2/400 Files (DUPPRODB)

Type choices, press Enter.

New Progress/400 DB Name . . . .  mydb
 Name, *CURLIB
From Progress/400 DB Name  . . .  *PROEMPTY
Character value, *PROEMPTY...
Create Schema Image  . . . . . .  *yes
 *YES, *NO
Case Sensitive ALTSEQ Table  . .  *NONE
 Name, *NONE
  Library  . . . . . . . . . . .    *LIBL
 Name, *LIBL
Case Insensitive ALTSEQ Table  .  *NONE
 Name, *NONE
  Library  . . . . . . . . . . .    *LIBL
 Name, *LIBL
Upper Case Table . . . . . . . .  *NONE
 Name, *NONE
  Library  . . . . . . . . . . .    *LIBL
 Name, *LIBL
Lower Case Table . . . . . . . .  *NONE
 Name, *NONE
  Library  . . . . . . . . . . .    *LIBL
 Name, *LIBL
Word Break Table Name  . . . . .  DFTWISWST
TBL Name or DFTWISWST for DFT
  Library  . . . . . . . . . . .    *LIBL
 Name, *LIBL
DataServer Database Code Page  .  *SYSVAL
Dictionary Library Description    *DFT
```

Figure 8–2: DUPPRODB Prompts and Responses

Note that in the example in Figure 8–2, the DataServer Database Code Page parameter is *SYSVAL, which specifies the AS/400 operating system code page. Specifying the AS/400 code page typically provides the best results. You will need a different code page only in very rare circumstances.

Once you have determined the code page to use and you know the default code page on your AS/400, you must choose the corresponding ALTSEQ table from the tables that IBM provides for the AS/400.

Table 8–3 provides a list of commonly used code pages that correspond to IBM ALTSEQ tables.

Table 8–3: Code Pages and Corresponding ALTSEQ Tables *(1 of 2)*

Code Page	Country or Alphabet	Case-sensitive ALTSEQ Table	Case-insensitive ALTSEQ Table	Uppercase ALTSEQ Table
037	USA	QLA10025U	QLA10025S	Q037
037	Canada	QLA10025U	QLA10025S	Q037
256	The Netherlands	QLA10100U	QLA10100S	Q256
273	Germany	QLA10111U	QLA10111S	Q273
273	Austria	QLA10111U	QLA10111S	Q273
277	Norway	QNOR0115U	QNOR0115S	Q277
277	Denmark	QDANR0115U	QDAN0115S	Q277
278	Sweden	QSNF0016U	QSNF0016S	Q278
278	Finland	QSNF0016U	QSNF0016S	Q278
280	Italy	QLA10118U	QLA10118S	Q280
284	Spain	QLA1011CU	QLA1011CS	Q284
285	United Kingdom	QLA1011DU	QLA1011DS	Q285
297	France	QLA10129U	QLA10129S	Q297
420	Arabia	QARA01A4U	QARA01A4S	Q420
870	Latin 2	QLA20366U	QLA20366S	Q870
1026	Turkey	QTRK0389U	QTRK0389S	QA3S
500	Denmark	QDAN01F4U	QDAN01F4S	Q500
500	Latin 1	QLA1014FU	QLA1014FS	Q500
500	Norway	QNOR01F4U	QNOR01F4S	Q500

Table 8–3: Code Pages and Corresponding ALTSEQ Tables *(2 of 2)*

Code Page	Country or Alphabet	Case-sensitive ALTSEQ Table	Case-insensitive ALTSEQ Table	Uppercase ALTSEQ Table
500	Spain	QESP01F4U	QESP01F4S	Q500
500	Sweden	QSNF01F4U	QSNF01F4S	Q500
500	Finland	QSNF01F4U	QSNF01F4S	Q500

Note that Table 8–3 does not include a list of lowercase tables. IBM does not provide lowercase tables, so you must build them yourself. You use lowercase tables only when the DataServer must evaluate the Progress 4GL function LC within a query. For information on building these tables, see the "Creating the Tables" section.

For more information about collation on the AS/400, or for a complete list of IBM ALTSEQ tables, see the IBM *AS/400 National Language Support Planning Guide*.

8.4.3 User-defined Collation Tables

This section describes how Progress/400 lets you define and use alternate collation sequences against your DB2/400 database. Specifically, it provides information on:

- What a user-defined collation table is
- Implementing user-defined collation tables
- Creating alternate collating sequence tables
- Limits and restrictions on collation tables

Collation sequences determine the order in which an application sorts data. A user-defined collation table lets you define and control how character data is collated (sorted). Because Progress/400 products use the native DB2/400 database, they are restricted to the collation capabilities provided by OS/400. The advantage of creating your own collation table is that you can define the collation sequence based on the national and cultural requirements specific to your application deployment environment. You can then use your custom collation table to control sorting, which allows you to collate character fields, whether indexed or not, in a user-defined order.

To take advantage of user-defined collation, you create a table and populate it with the hexadecimal value for each character, assigning the values in the order in which you want character data to sort. You can also sort with tables provided by the OS/400, located in the QUSRSYS library. A second table, called the *uppercase table*, controls lowercase-to-uppercase character conversion. You use a table to look up a character and its corresponding numeric value.

Implementing User-defined Collation Sequences

To use an alternate collation sequence, Progress/400 requires that you specify one of the alternate collating sequence tables that your system provides. If your system does not provide one that addresses your collation requirements, you can create your own.

The Progress/400 DataServer works with four tables. These tables are objects of the *TBL data type that either come with the system or you define:

- Alternate collating sequence table (ALTSEQ)

  ALTSEQ must be a DB2/400 table whose source is a source member that contains eight records, each of which contains 64 hexadecimal digits. (The 8 x 64 format is required by DB2/400.) The 64 digits represent 32 values, from 0-225 (xFF). The total of 8 records provide collation values for all 256 EBCDIC characters. Characters are sorted in the order of their collation values.

- Alternate collating sequence case-insensitive table (ALTSEQCI)

  ALTSEQCI is a DB2/400 table whose source is a source member that contains eight records, each of which contains 64 hexadecimal digits. (The 8 x 64 format is required by DB2/400.) The 64 digits represent 32 uppercase characters that correspond to the 32 characters represented by that record. This table ensures Progress-like behavior for case-insensitive indexes.

- Progress uppercase table (UPCASE)

  UPCASE is a DB2/400 table. The source for the table must be a source member that contains eight records, each of which contains 64 hexadecimal digits. (The 8 x 64 format is required by DB2/400.) The 64 digits represent 32 uppercase characters that correspond to the 32 characters represented by that record.

- Progress lowercase table (LOCASE)

  LOCASE is a DB2/400 table. The source for the table must be a source member that contains eight records, each of which contains 64 hexadecimal digits. (The 8 x 64 format is required by DB2/400.) The 64 digits represent 32 lowercase characters that correspond to the 32 characters represented by that record.

To use an alternate collation sequence, you must first build the collation-sequence and uppercase/lowercase tables.

When you install the Progress/400 DataServer, Progress allows you to select a case-insensitive table as the default. When you create a dictionary library, the DUPPRODB utility allows you to override this default table. Each time you connect to your DB2/400 database, Progress looks for the collating and uppercase tables you specified when you created the Progress/400 Dictionary and server schema with the DUPPRODB utility. If the tables exist, Progress reads and writes data based on that information. If they do not exist, Progress uses the default collation sequence.

Creating the Tables. You use the OS/400 Create Table (CRTTBL) command. This is its syntax:

SYNTAX

```
CRTTBL TBL(library/table-name) SRCFILE(library/table-source-file)
 SRCMBR(*TBL) TEXT(description) AUT(authority)
```

The CRTTBL command expects the source member (SRCMBR) to contain eight records of 64 hexadecimal characters each. An example of a collation table is shown in Figure 8–3.

```
000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F
20212223242526...          ...3E3F
C16362C2...
60...
80...
A0...
C0...
E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEFF0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF
```

Figure 8–3: Collation Table Example

Figure 8–4 illustrates two points. Each point is highlighted by a number to the left; the notes that follow Figure 8–4 describe their collation functions.

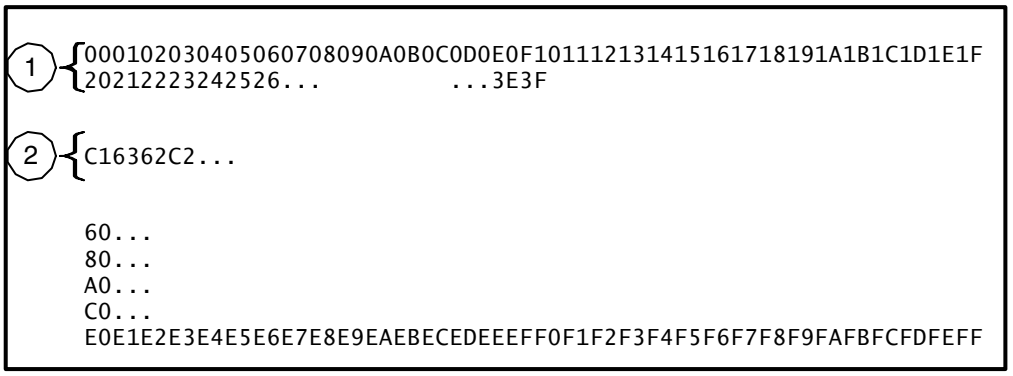


Figure 8–4: Specifying a Sort Order in a Collation Table

1. The first two lines of EBCDIC collation tables provide the collation sequence for the unprintable characters (the characters between 00 and hex3F).

2. This line illustrates how to use the hexadecimal values from code page IBM500 to sort the following characters in the order shown: A Ä, Â, B.

You create an uppercase table in the same way you create a collation table, except that in the places that contain lowercase characters, you substitute the hexadecimal value of the equivalent uppercase. For example, in the table shown in Figure 8–4, instead of x81 you place an xC1 so that the fifth line starts with 80C1C2.

Collation Table Limits and Restrictions

There are several limits and restrictions to consider when defining collation tables:

- Progress has a default collation sequence and a default lowercase-to-uppercase table.
- If you do not specify a specific collation sequence, the default is the standard collation sequence for your AS/400.
- When you create an alternate collation sequence, the OS/400 *TBL object requires that you define a collation sequence for all 256 characters in a code page regardless of whether you use them all.
- If your user-defined collation table is not set up correctly, you might receive unpredictable results.

DataServer Considerations

This section explains two important issues for defining alternate collation sequences for use with the Progress/400 DataServer:

- Using the correct translation tables
- Using the proper hexadecimal values for sorting uppercase characters

Conversion Tables for User-defined Collation. The Progress/400 DataServer provides international character support using a file called `CONVMAP.DAT`. This file resides on the Progress client and consists of entries for ASCII and EBCDIC Code Pages for various languages.

When the Progress client connects to the server on the AS/400, the client loads the two tables required for the translation from the `CONVMAP.DAT` file into memory. For example, if the client uses ISO8859-1 and the server uses IBM037, the client loads one table to handle the conversion from ISO8859-1 to IBM037 and a second table to handle the conversion from IBM037 to ISO8859. The client uses the tables to convert its requests into the code page required by the AS/400 server. When the server fulfills and returns the database request, the client uses `CONVMAP.DAT` to convert between the AS/400 code page and the client code page. The client handles all code-page conversions. It converts only those fields required by the Progress application.

8.5 Controlling DataServer Performance

There are two general areas to consider when fine-tuning the DataServer environment for performance:

- You can build queries that take advantage of the strengths of client/server or n-tier processing.
- You can consider optimizing how database results are transmitted through network connections

The following sections provide some guidelines for making both types of adjustments.

8.5.1 Query Processing

By default, the Progress/400 DataServer instructs the server to evaluate a query. This can enhance performance depending on whether your system is configured to maximize server processing abilities or network overhead.

To take advantage of the server's query processing, use FOR EACH statements combined with GET NEXT instead of FIND and REPEAT combinations when programming in the Progress 4GL.

If selection by server is not optimal for your configuration, you can use the DataServer (-Dsrv SBS=0) startup parameter to prevent it and allow the client to evaluate database results.

8.5.2 Managing Network Traffic

Fine-tuning how database results are packaged and communicated between the server and client can enhance performance for DataServer applications. Two Progress parameters allow you to control the size of a network message and the contents of the message:

- Message Buffer Size (-Mm) startup parameter
- DataServer (-Dsrv COMPRESS) connection parameter

You can set the -Mm parameter to take full advantage of the DataServer's default prefetch behavior, which packs as many records as possible in one message. For example, if you have 500 byte records, the DataServer packs two records into one message. (The default Progress message buffer size is 1 Kb.)

Prefetch and -Mm combined with the -Dsrv COMPRESS=1 parameter can enhance performance even more. The COMPRESS option can compress records up to 50%. In the example, the DataServer can compress the 500 byte records by 50% so that it can pack four records into one message.

8.5.3 Loading Definition Files

Progress optimizes the loading of definition files if you create a dictionary library with DUPRODB FRMDB(*PROEMPTY) and have no other database files defined in the server schema at load time. This method offers the optimal performance for loading definition files into an empty dictionary library on the AS/400.

8.6 Progress Settings File (PROSET)

Progress/400 provides a settings file, called PROSET, that regulates its behavior. This file is located in the Progress library as supplied in the product media. You can change the settings using various OS/400 utilities, including the Data File Utility (DFU).

NOTE: **Do not** modify any setting that is not documented.

Table 8–4 shows the settings for PROSET and the effects of assigning different values to those settings.

Table 8–4: PROSET Settings *(1 of 5)*

Setting Name	Description
QUEUE-DUPLICATE-SERVER-MESSAGES	Determines whether the server sends duplicate messages to the client when requested when an error occurs. Specify YES or Y (the default) to send duplicate messages. Specify NO or N to not send duplicate messages. Program/Utility: General
COMMITMENT-CONTROL-SCOPE	When commitment control is started, it is started based on this value. Specify *ACTGRP to start commitment control for the current activation group only. In most cases, this covers the currently running Progress Server or native clients. Any programs called using either QCMD or EPI usually start their own activation group. Specify *JOB to start commitment control for the entire job. Any called programs are included in the commitment definition started for the server or the native client. You can regulate the effect of this switch for the client-server environment (PROSERV) and the native environment separately. To do this, enter a record into the PROSET file using each of the two key values. Program/Utility: PROCLIENT, PROSERV

Table 8–4: PROSET Settings *(2 of 5)*

Setting Name	Description
ALLOW-ZONE-DECIMAL	Determines whether to allow the use of ZONED decimal numbers with decimal digits. Specify YES or Y to allow the use of ZONED decimal numbers with decimal digits. The actual number of decimal digits is placed into the DDS. Specify NO or N (the default) to not allow the use of ZONED decimal numbers with decimal digits. ZONED decimal numbers are created with zero (0) decimal digits. For more information, see the "ALLOW-ZONE-DECIMAL Usage Notes" section. Program/Utility: CHGPRODCT
MAP-VARIANT-CHARACTERS	Determines whether variant characters (#,@,$) are mapped to underscore (_). Specify YES or Y (the default) to map variant characters. Specify NO or N to not map variant characters. However, if a variant character is the first character, map it to "A". Program/Utility: CHGPRODCT
ALLOW-SELECT-OMIT-INDEXES	Determines whether to allow Select/Omit Logical files as indexes. Specify YES or Y (the default) to allow Select/Omit Logical files as indexes. Specify NO or N to not allow Select/Omit Logical files as indexes. Program/Utility: CHGPRODCT

Table 8–4: PROSET Settings *(3 of 5)*

Setting Name	Description
USE-CHGPF	Specify whether APYPRODCT will delete and re-create physical files as in prior versions or use the OS/400 CHGPF command. Specify Yes or Y to use the CHGPF command. Specify No or N (the default) to use existing methods. Program/Utility: APYPRODCT
PROCESS-DFT-DDS-KEYWORD	Allows the user to regulate the processing of the DDS keyword DFT. This keyword sets up a DEFAULT value for the field. This DEFAULT value becomes the Progress Initial Value under the following conditions: - If the field is numeric, the DFT value is placed directly into the Progress initial field. - If the field is character, the DFT value is stripped of quotation marks and then placed directly into the Progress initial field. When HEX values or *NULL have been specified in the DFT keyword, the Progress initial value is not loaded. Specify YES or Y to process the keyword and place it into the Progress initial field. Specify NO or N (the default) to ignore the DFT DDS keyword. Program/Utility: CHGPRODCT

Table 8–4: PROSET Settings *(4 of 5)*

Setting Name	Description
EXCLUDE-NULL-KEY-VALUE	Allows you to have more than one record with the UNKNOWN value in the index field when the index is defined as a UNIQUE index. It excludes the UNKNOWN/NULL index value when determining duplicate key values for a UNIQUE index. Specify YES or Y to add the (*EXCNULL) parameter to the UNIQUE keyword in the DDS of the file on the AS/400. This excludes null values when determining duplicate index values. Specify NO or N (the default) to include the NULL value in determining duplicate index values. Program/Utility: APYPRODCT
CHANGE-CHAR-TO-VARLEN	Allows the user to have character fields changed to character variable-length fields during the commit process if the length of a field is greater than the length specified in the VALUE setting of the PROSET setting. Specify an integer value greater than 0 to enable this change. For example, if you specify ’200’, a character field is changed to a character variable-length field if its length is greater than 200. Specify 0 to not change character fields to character variable-length fields. Program/Utility: APYPRODCT

Table 8–4: PROSET Settings *(5 of 5)*

Setting Name	Description
INCLUDE-VIRTUAL-FILES	Determines whether CHGPRODCT processes virtual files. Specify Y (the default) to process virtual files. Specify N to not process virtual files. Program/Utility: CHGPRODCT
INCLUDE-SOURCE-FILES	Determines whether CHGPRODCT processes source files. Specify Y (the default) to process source files. Specify N to not process source files. Program/Utility: CHGPRODCT

NOTE: If variant characters are not mapped, it is possible to create a file or field name that is invalid to Progress/400. For example, the pound sign (#) is a valid character for OS/400 but invalid to Progress/400.

See your *Progress/400 Release Notes* for additional settings that you can add to your PROSET file.

ALLOW-ZONE-DECIMAL Usage Notes

This section describes how the ALLOW-ZONE-DECIMAL setting in the PROSET file affects files when you use the Progress/400 Data Dictionary to create or modify files.

Setting ALLOW-ZONE-DECIMAL to YES has the following effects:

- On new files created with the Progress/400 Data Dictionary, the setting has no effect.
- When you use the Progress/400 Data Dictionary to maintain an existing file that was added with the Progress/400 Data Dictionary and the file contains data, then when the data is copied back into the file and decimals are added, the data is shifted to the left by the number of decimals.

For example, suppose that the Balance field of a record in the customer file in the Progress-supplied PRODEMO sample database contains the value 93545, then you perform maintenance on the file (not necessarily to this field). After performing maintenance, the value in the Balance field is 9354500.

This change happens whenever you perform an action in the Progress/400 Data Dictionary that results in the regeneration of the DDS. This means that if a user adds or deletes a field and does not touch the zoned field, the value in the zoned field still changes.

If you want to set ALLOW-ZONE-DECIMAL to Y but ensure that data values remain correct when you use the Progress/400 Data Dictionary, you must do the following:

a) Before making a change, dump the data.

b) After making the change, delete all records.

c) Reload the data.

- When you either use the Progress/400 Data Dictionary to maintain an existing file that was originally added with CHGPRODCT or perform maintenance using CHGPRODCT, the file maintains the proper decimals and the data values are correct.

Setting ALLOW-ZONE-DECIMAL to NO has the following effects:

- On new files created with the Progress/400 Data Dictionary, the DDS does not have decimals.
- When you use the Progress/400 Data Dictionary to maintain an existing file that was originally added with the Progress/400 Data Dictionary, there are no behavioral changes and the data values remain the same.
- When you use the Progress/400 Data Dictionary to maintain an existing file that was added with CHGPRODCT, the DDS loses the decimals and any data is truncated.

8.7 Retaining Progress Attributes

You can apply database changes on the AS/400 with the Change Progress/400 Dictionary Utility (CHGPRODCT) and retain attributes of the files from the original Progress/400 dictionary creation. To retain Progress-style field names, you must specify the correct value for the PROATR keyword in CHGPRODCT.

This feature does not apply in the following situations:

- A file has extent fields.
- A file has changed; for example, from a logical file to a physical file.
- A Progress/400 field data type has changed (character, integer, and so forth).
- It is possible that the index designated as primary might not be retained. It is affected by the value of the PROATR option. If *CMPSAVE is selected, Progress issues a message if the value has changed.
- An initial value was specified for a field on the AS/400. CHGPRODCT always overwrites the initial value set from the Progress/400 Data Dictionary.

Table 8–5 shows the affect of PROATR on fields given the three possible keyword selections.

Table 8–5: CHGPRODCT PROATR Behavior *(1 of 2)*

Progress Field	*OVRWRT	*SAVE	*CMPSAVE
FILE RELATED:			
DESCRIPTION	Generated	Current value	DDS value
FILENAME	DDS Value (FILE)	Current value	Current value
DUMPNAME	Generated	Current value	Current value
FILE LABEL	Blank	Current value	Current value
FILE LABEL SA	Blank	Current value	Current value
HIDDEN	Defaults to “N”	Current value	Current value
VAL EXP	Blank	Current value	Current value
VAL EXP SA	Blank	Current value	Current value
PRIME INDEX	Generated (first index)	Generated	Generated (message if changed)
FIELD RELATED:			
FIELDNAME	Generated (AS4FIELD)	Current value	Current value

Table 8–5: CHGPRODCT PROATR Behavior *(2 of 2)*

Progress Field	*OVRWRT	*SAVE	*CMPSAVE
DESCRIPTION	Generated/Blank	Current value	DDS value
COL LABEL	DDS value	Current value	DDS value
LABEL	DDS value	Current value	DDS value
COL LABEL SA	Blank	Current value	Current value
DECIMALS	Generated	Current value	Current value
FORMAT	Generated	Current value	Current value-validate
FORMAT SA	Blank	Current value	Current value
HELP	Blank	Current value	Current value
HELP SA	Blank	Current value	Current value
VAL EXP	Blank	Current value	Current value
VIEW AS	Blank	Current value	Current value
VAL MSG	Blank	Current value	Current value
INDEX RELATED:			
INDEXNAME	Generated	Current value	Current value
DESCRIPTION	Generated	Current value	DDS value
ACTIVE	Defaults to “Y”	Current value	Current value

For more information on CHGPRODCT, see Chapter 10, “AS/400 Utilities.”

8.8 Progress/400 Reserved Words

You can find a list of reserved Progress/400 words in a file named PRORSVWRD that exists in the OS/400 Progress library. You can modify this list by using standard OS/400 utilities such as the Data File Utility (DFU).

8.9 Creating Test and Production Environments

The Progress/400 Duplicate Dictionary (DUPPRODB) utility allows you to duplicate existing Progress/400 dictionaries. The user performing DUPPRODB operations should have *OBJEXIST, *OBJMGT, *OBJOPR, and *READ authority to all files in the source dictionary.

NOTE: DUPPRODB is the only technique you can use on the AS/400 to duplicate existing Progress/400 server schemas. To access database files, the Progress/400 DataServer uses server schema information to find where each file resides. Do not use OS/400 CL commands, such as backup/restore and copy, to copy the server schema.

The DUPPRODB process follows:

1. DUPPRODB creates a new object library and a new server schema. The object library is the destination for all the files that you want to duplicate.

2. DUPPRODB copies the physical and logical files from the existing server schema that you specify into the new server schema and object library.

There are several options for how DUPPRODB duplicates the server schema. You determine the way the DB2/400 database files are duplicated by your entry at the Duplicate Files (CPYOPT) parameter:

- The *FULLCPY option allows you to duplicate all of the files (including data) that are part of the server schema, regardless of the library where the files reside. This option creates an exact duplicate of the existing server schema, except that each duplicate file resides in the new library. Previously, the files might have resided in several libraries.

  Use this option when you want to make an exact duplicate of a database so you can use one database for testing and one for production.

  CAUTION: This command does not duplicate all objects in a library. It duplicates only the objects in a library that are defined in a server schema.

- The *EMPTYCPY option operates identically to the *FULLCPY option except that it does not copy the data that the objects contain; that is, the new objects are empty.

- The *DCTONLY option allows you to duplicate only the server schema.

See Chapter 10, “AS/400 Utilities,” for a detailed description of DUPPRODB.

Figure 8–5 and Figure 8–6 illustrate the use of the *FULLCPY and *DCTONLY options.

- Figure 8–5 shows the results you get when you use DUPPRODB with the *FULLCPY option, as in the following command:

```
DUPPRODB NEWDB(MYSPORTS) FRMDB(PROSPORT)
  CPYOPT (*FULLCPY) OVRWRTDCT(*YES) CRTLKT(*NO)
```

  Figure 8–5 also applies to using DUPPRODB with the *EMPTYCPY option **except that** the data files contain no data.

- Figure 8–6 shows the results you get when you use DUPPRODB with the *DCTONLY option, as in the following command:

```
DUPPRODB NEWDB(MYSPORTS) FRMDB(PROSPORT)
  CPYOPT (*DCTONLY) OVRWRTDCT(*YES) CRTLKT(*NO)
```

See Appendix A, "Tutorials for Managing Your Dictionary Library," for more examples on duplicating a Progress database.

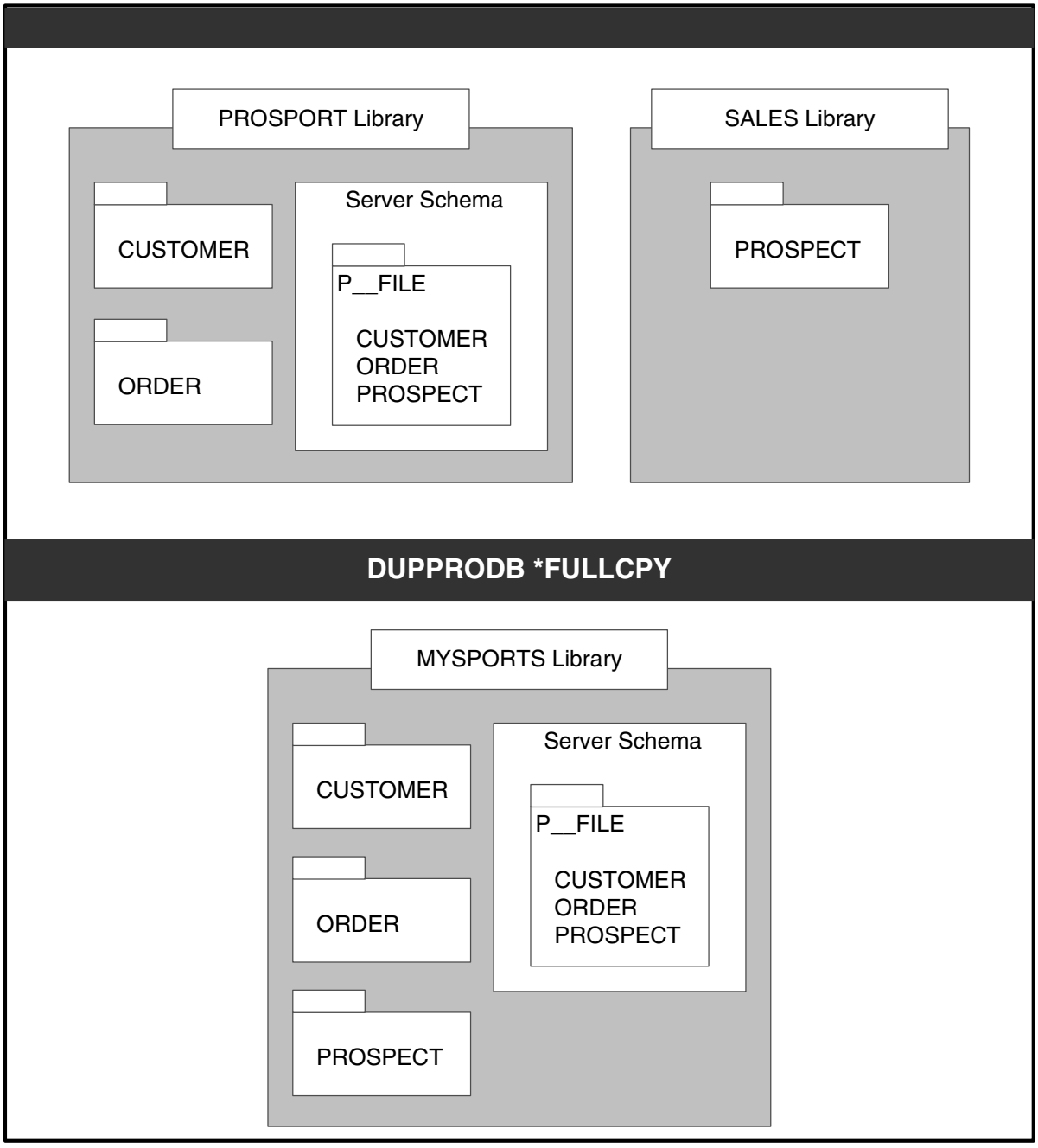


Figure 8–5: How DUPPRODB Works with the *FULLCPY Option

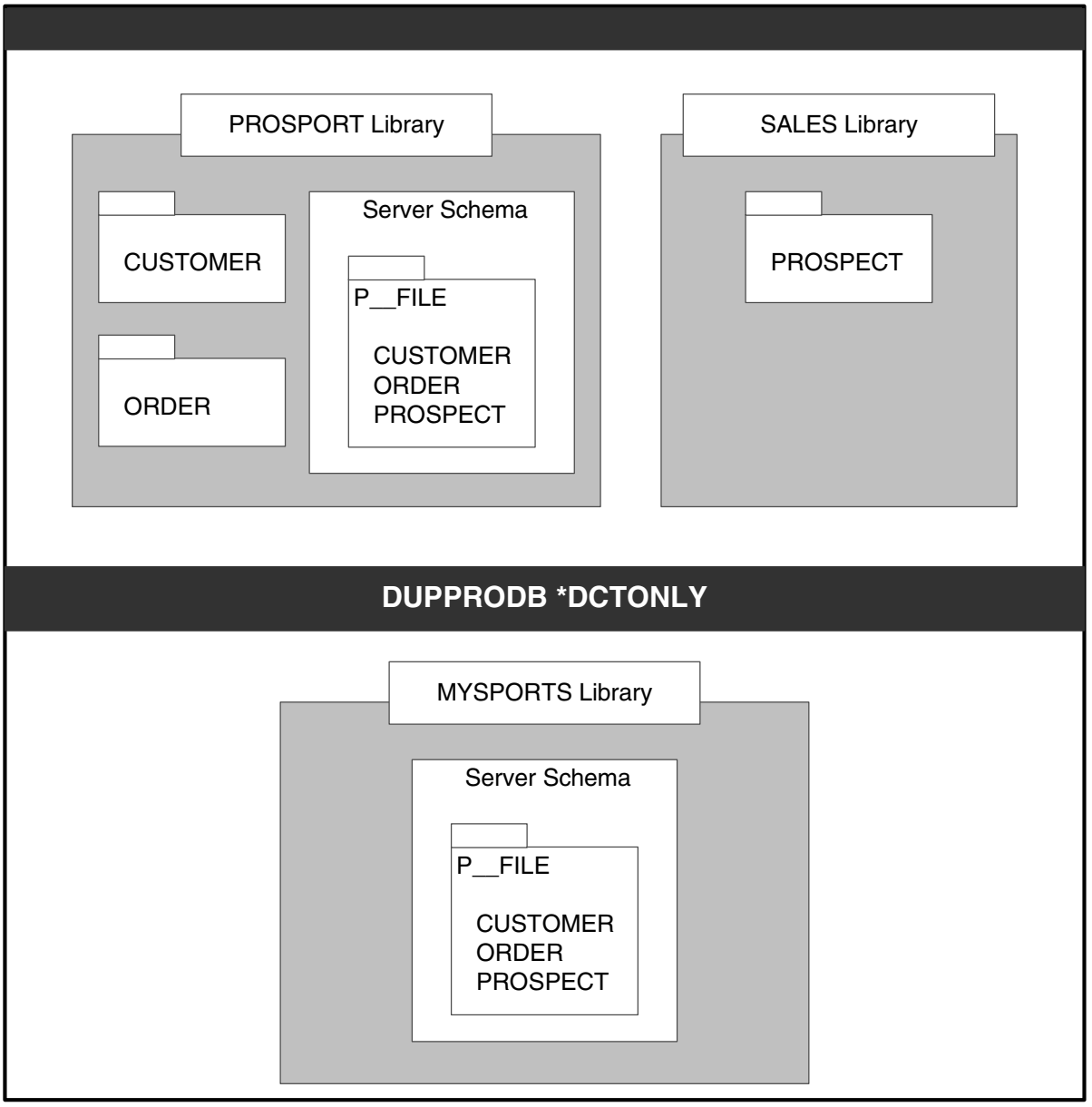


Figure 8–6: How DUPPRODB Works with the *DCTONLY Option

9

Remote Client DB2/400 Utilities

This chapter describes the client utilities and the tasks associated with them. It covers the following tasks:

- Using the DataServer utilities to maintain client schema holders
- Using the Progress/400 Data Dictionary to maintain DB2/400 data definitions
- Using the Progress/400 Data Dictionary database administration utilities

9.1 DataServer Utilities in the Data Administration Tool

Table 9–1 describes the utilities for the Progress/400 DataServer. These utilities are available from the Data Administration main menu. The following sections introduce you to the various utilities.

Table 9–1: DataServer Utilities

DB2/400 Utility	Description
Progress/400 Data Dictionary	Use this tool to maintain data definitions on the AS/400. It is available on MS-Windows only.
Synchronize Progress/400 Client...	Use this utility to update the schema holder to reflect any changes made to DB2/400 data definitions and run the verify option.
Edit DB2/400 Connection Information...	Use this utility to change connection information for a DB2/400 database.
Create DB2/400 DataServer Schema...	Use this utility to create a client schema holder for a DB2/400 database.
Delete DB2/400 DataServer Schema...	Use this utility to delete a schema holder.

9.1.1 Create DB2/400 DataServer Schema

This utility creates a client schema holder. You must connect to a Progress database before using it. See Chapter 3, "Creating the Progress/400 Environment," for instructions on this utility when either accessing an existing DB2/400 database or moving a Progress database to the AS/400.

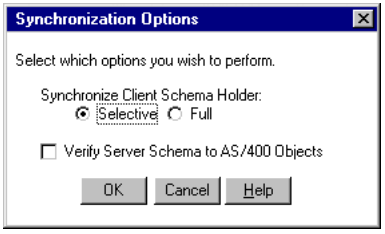
9.1.2 Synchronizing a Client Schema Holder

This utility synchronizes the client schema holder with the server schema on the AS/400. You must synchronize the client schema holder whenever you modify data definitions in your server schema to make sure that your client schema holder accurately reflects the changed information.

Follow these steps to synchronize a client schema holder:

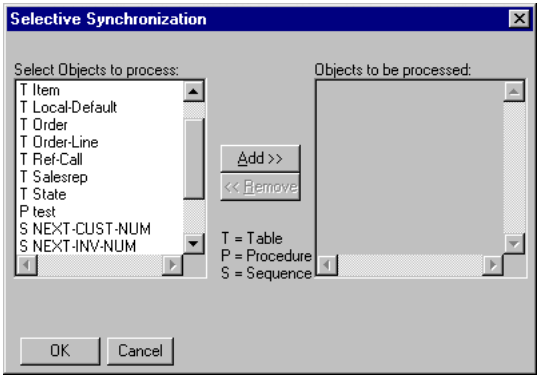
1 ♦ Access the Data Administration tool.

2 ♦ Choose DataServer→DB2/400 Utilities→Synchronize Progress/400 Client. The following dialog box appears:



3 ♦ Choose the Selective or Full radio button to perform a selective or a full synchronization.

If you choose selective synchronization, the following dialog box appears:



This window displays the files that you can select for synchronization.

4 ♦ If you chose the Selective button, choose the objects to synchronize:

- To select an object, use the up and down arrow keys in the "Select Objects to process:" pane to highlight a name, then press the RETURN key or the Add >> button to select it. The name moves to the Objects to be processed: pane. Repeat this process for each object to be synchronized.

- To remove a selected object, use the up and down arrow keys in the "Objects to be processed:" pane to highlight the object name, then press the **RETURN** key or the << Remove button to remove it. The object name moves to the Select Object to process: pane. Repeat the selection process for each object to be removed.

After your selection list is complete, choose OK.

5 ♦ Choose Verify Server Schema to verify the physical file objects against the information in the server schema. If the utility finds any differences, it displays a report.

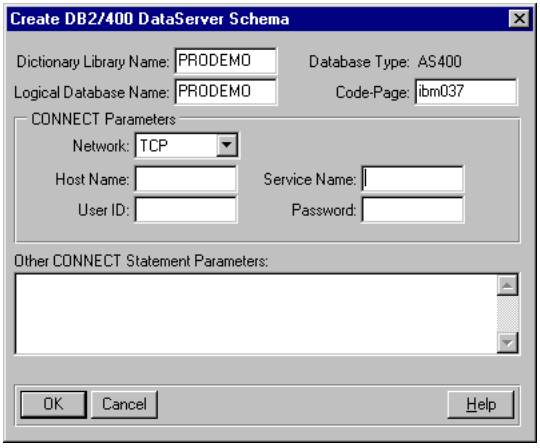
6 ♦ Choose OK.

You can also access this utility from the Admin menu in the Progress/400 Data Dictionary. In addition, the installation media includes the `as4sync.p` procedure that you can run in an application to synchronize deployed schema holders.

9.1.3 Changing Connection Information

Follow these steps to change connection information for a DB2/400 database:

1 ♦ Choose DataServer→ DB2/400 Utilities→ Edit Connection Information. The following dialog box appears:



2 ♦ Modify the Connection Parameters. When you are done, choose OK to return to the main window.

The changes do not take effect until you disconnect and reconnect the client schema holder. When you reconnect, Progress uses the new connection parameters.

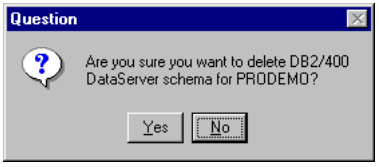
For details on connection parameters, see Chapter 4, "Remote Access to Progress/400 DataServer," and the *Progress Startup Command and Parameter Reference*.

The Logical Database Name can be changed using this utility. If the DB2/400 database is connected, it will be disconnected and reconnected using the new logical name. If the DB2/400 database is not connected, then the logical name is changed and the tool you are working on is closed. This is necessary so the proper aliases are reassigned to the new logical name. When the tool is reselected, the new logical name will be in effect.

9.1.4 Deleting a Client Schema Holder

Follow these steps to delete a schema holder:

1 ♦ From the Data Administration main window, choose DataServer→ DB2/400 Utilities→ Delete DB2/400 DataServer Schema. The following dialog box appears:



2 ♦ Choose Yes to verify your selection. The utility deletes the schema from the client schema holder.

This utility deletes the schema information from the client schema holder. It does not delete the database that serves as the schema holder or any user data in the DB2/400 database. For example, if you have the schema for three non-Progress databases stored in a schema holder such as DB2/400, ORACLE, or ODBC, you can delete the DB2/400 schema information and leave the other schema information intact.

9.2 Progress/400 Data Dictionary

The Progress/400 Data Dictionary is a graphical environment on the client that allows you to modify DB2/400 data definitions and perform administrative functions on DB2/400 database files. Based on the standard Progress Data Dictionary, the Progress/400 Data Dictionary includes the same features. In addition, it has utilities for dumping and loading data and data definitions. You can use the Progress/400 Data Dictionary to modify, add, or delete tables, fields, indexes, sequences, and stored procedures.

The Progress/400 Data Dictionary is fully integrated into the Progress Application Development Environment (ADE). You access it from the DataServer menu in the Data Administration tool.

Figure 9–1 shows the Progress/400 Data Dictionary window.



Figure 9–1: Progress/400 Data Dictionary Window

The Progress/400 Data Dictionary allows you to define Progress features that are not supported by DDS, such as field labels, display formatting, triggers, validation expressions, validation messages, help strings, arrays, data types, and case-insensitive indexing. You can also provide descriptions (up to 50 characters long) of objects.

NOTE: If DB2/400 database files contain DDS definitions that the DataServer does not support, do not use the Progress/400 Data Dictionary to maintain or modify them. Continue to maintain the database files through DDS. You can use the Progress/400 Data Dictionary to view the supported subset of data definitions and generate reports on them.

The Progress/400 Data Dictionary supports word indexes, generally in the same manner as the Progress Data Dictionary does. Minor differences are noted, where relevant, in this chapter. For general information on word indexes, see the *Progress Programming Handbook*.

Before you can use the Progress/400 Data Dictionary to access DB2/400 database objects, you must create a Progress/400 Dictionary Library on the AS/400 that includes the server schema for those objects. See Chapter 3, "Creating the Progress/400 Environment," for detailed instructions.

The Progress/400 Data Dictionary operates in two modes:

- **Modify schema** — This mode requires that the client session connect to the AS/400 server as Database Administrator (DBA) and that the current user have authority to execute the Change Journal (CHGJRN) CL command. When you access DB2/400 database files in DBA mode, the OS/400 locks the objects and prevents other users or jobs from accessing them. Use this mode only when you plan to make changes to DB2/400 data definitions.

- **Read only** — This mode allows you to view DB2/400 data definitions; load data; and generate table, field, index, and sequence reports. It does not lock the server schema.

You must synchronize the client schema holder with the server schema before performing any tasks with the Progress /400 Data Dictionary that involve dumping or loading data. You do not have to synchronize before modifying, dumping, or loading data definitions.

Alternate between modify-schema mode and read-only mode by activating the appropriate radio button in the Progress/400 Data Dictionary main window.

Be sure to commit changes that you make when you move from modify-schema mode to read-only mode in the Progress/400 Data Dictionary. If you select not to commit changes, the DataServer rolls back any uncommitted changes.

9.2.1 Adding a Table

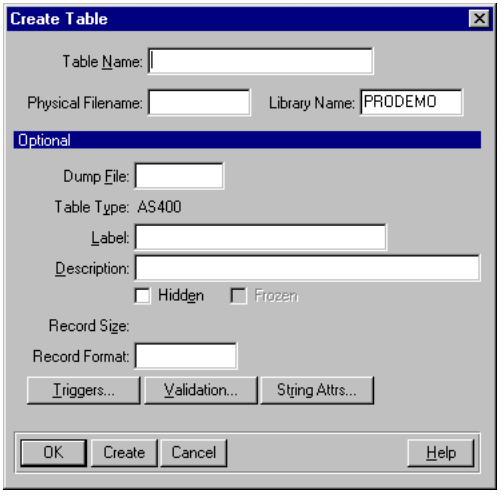
The Progress/400 Data Dictionary allows you to add a table to an existing library. The DataServer takes the definitions you provide and creates a physical file on the AS/400 with those characteristics. You can then access that physical file through Progress applications or through AS/400 applications. Non-Progress applications ignore Progress features that you might include, such as validation expressions and messages.

Follow these steps to add a table:

1 ♦ Choose the Table mode button in the Progress/400 Data Dictionary main window.

2 ♦ Choose Modify schema.

3 ♦ Choose the Create Table button. The following dialog box appears:



4 ♦ Enter a name in the Table Name field. This is the name by which Progress recognizes this object, so it should follow Progress naming conventions. Specify a unique name.

5 ♦ Press **TAB** and the physical file, library, and dump filenames are supplied for you based on the table name. For example, for a table named newcust, the AS/400 physical file name is NEWCUST, the library name is the object library.

If you do not want default values, enter the information in each field.

6 ♦ Enter other information that you want to add to the definition for the new table. Choose the Help button to access detailed information on all of the elements in the dialog box.

7 ♦ Choose OK.

8 ♦ Create fields and indexes for the file.

You modified the data definitions in the server schema on the AS/400. Your server schema now contains an object that is not reflected in the client schema holder until you synchronize it.

9.2.2 Modifying a Table

The Progress/400 Data Dictionary allows you to modify DB2/400 database files, except for files with multiple members.

CAUTION: The DataServer changes the definition of the physical file with the new information. Any DDS definitions in the physical file that are not supported by the DataServer will be lost.

The Progress/400 Data Dictionary preserves DB2/400 file attributes with the exceptions of those described in Table 9–2.

Table 9–2: DB2/400 File Attributes

Attribute	Exception
DTAMBRS (logical file)	Changes with the CRTLF command defaults
IGCDATA (physical file)	Defaults to *NO
MBR (physical and logical file)	Defaults to *FILE
SHARE (physical file)	Changes with the CRTPF command defaults

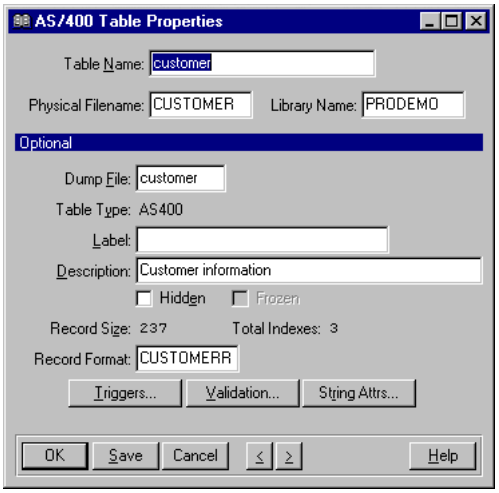
Follow these steps to modify a table:

1 ♦ Choose the Table mode button in the Progress/400 Data Dictionary main window.

2 ♦ Choose Modify schema.

3 ♦ Select a table from the Tables list.

4 ♦ Choose the Table Properties button. The following dialog box appears:



5 ♦ Modify the fields to change the definitions for the DB2/400 file. Choose the Help button to access detailed information on all of the elements in the dialog box.

6 ♦ Choose OK to return to the Progress/400 Data Dictionary.

You modified the data definitions in the server schema on the AS/400. Your server schema now contains definitions that are not reflected in the client schema holder until you synchronize it.

9.2.3 Deleting a Table

The Progress/400 Data Dictionary allows you to delete a table in the server schema. It also gives you the option of deleting the associated DB2/400 physical file. If you choose to delete the table only from the server schema, you can still access the associated database file through non-Progress AS/400 applications. However, if you choose to delete the physical file, no application can access it. When you delete a physical file, any logical files associated with it are deleted, regardless of whether other physical files refer to the logical files. For example, if a join

logical file refers to the Customer and Order tables, and you delete the Order table and do not preserve the DB2/400 physical file, the join logical file is deleted also.

Virtual files are an exception. The Progress/400 Data Dictionary allows you to delete these from the server schema. However, it does not allow you to delete the associated DB2/400 logical files.

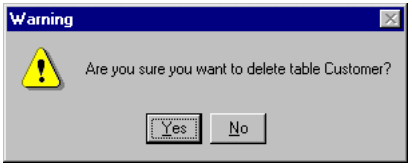
Follow these steps to delete a table:

1 ♦ Choose the Table mode button in the Progress/400 Data Dictionary main window.

2 ♦ Choose Modify schema.

3 ♦ Select a table from the Tables list.

4 ♦ Choose the Delete Table button. The following message appears:



5 ♦ Choose Yes.

6 ♦ A message appears asking you whether you want to delete the DB2/400 physical file.

7 ♦ Choose Yes to delete the physical file, or choose No to delete the definition from the server schema only and preserve the DB2/400 physical file.

You modified the data definitions in the server schema on the AS/400. Your server schema now has changes that are not reflected in the client schema holder until you synchronize it.

9.2.4 Adding a Field

The Progress/400 Data Dictionary allows you to add fields to existing DB2/400 files. You can then access that field through Progress applications or through AS/400 applications. Non-Progress applications ignore Progress features that you might include, such as validation expressions and help strings.

NOTE: You can change all of the attributes of a field in the Field Properties dialog box but **only before you commit the file**. Once you commit it, the Progress/400 Data Dictionary restricts what attributes you can change.

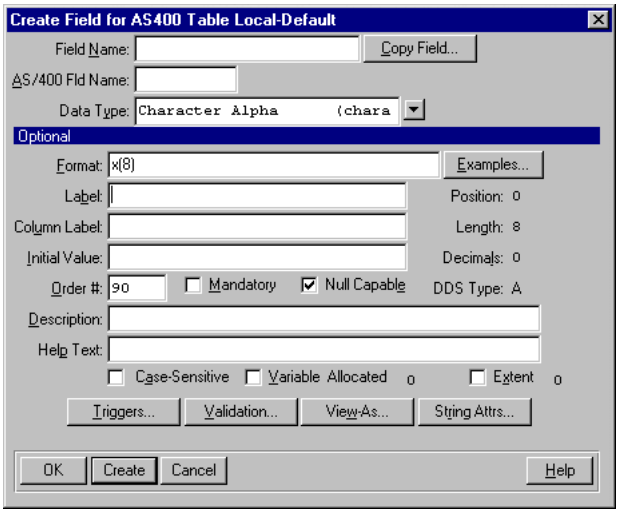
Follow these steps to add a field:

1 ♦ Choose the Field mode button in the Progress/400 Data Dictionary main window.

2 ♦ Choose Modify schema.

3 ♦ Select the table from the Tables list that will contain the new field.

4 ♦ Choose the Create Field button. The following dialog box appears:



5 ♦ Enter a name in the Field Name field. This is the name by which Progress recognizes this object, so it should follow Progress naming conventions. See the "Naming Conventions" section in Chapter 2, "Common Product Information."

6 ♦ Press **TAB** and the AS/400 name is supplied for you based on the field name. For example, for a field named ncust-num, the AS/400 physical field name is NCUST_NUM.

If you do not want a default AS/400 field name, enter the information.

7 ♦ Choose a data type. A default format is supplied based on the data type. You can change the default format. The format determines the size of character fields created in the physical file.

8 ♦ Enter other information that you want to add to the definitions for the new field. Choose the Help button to access detailed information on all of the elements in the dialog box.

9 ♦ Choose OK to return to the Progress/400 Data Dictionary.

You modified the data definitions in the server schema on the AS/400. Your server schema now contains definitions that are not reflected in the client schema holder until you synchronize it.

9.2.5 Defining Field Length

When defining fields in tables using the Progress/400 Data Dictionary field properties panel, the size of the field is calculated from the format that you specify for the field when you add it. If the calculated size is larger than the AS/400 DDS maximum size, it is truncated to the maximum. See the *AS/400 DDS Guide* for maximum lengths of the various data types.

Once you commit field changes, the only way to change the field size is to delete the field and then add it again. If you change a field's name, committing that change will fail. Again, you must delete the field and add it with the new name.

CAUTION: Whenever you delete a field, the data in the field is lost. Although you can successfully change a field's name, data type, or size by deleting and then adding it, the data in the field is lost.

9.2.6 Field Format Length

The Progress/400 server schema allows storing only 30 characters of field format mask. If you define a field as having 25 digits with 5 decimal positions using DDS on the AS/400 and then add the file to the Progress/400 server schema with the CHGPRODCT utility, the Progress/400 Data Dictionary Field Properties show 23 digits with 2 decimal positions, including commas and the decimal point.

To see all the digit positions, you must change the display format for the field using the Progress/400 Data Dictionary and remove the comma separators from the format to allow more room for all the digits. You can also change your code to display the proper format mask.

Even though the Progress/400 Data Dictionary Field properties sheet allows for 48 format positions, only 30 can be stored in the Progress/400 server schema.

If you exceed the 30-character limit, you receive an error message that the field has been truncated because of an overflow.

9.2.7 Modifying a Field

The Progress/400 Data Dictionary allows you to modify fields in DB2/400 database files. The DataServer takes the definitions you provide and adds them to the physical file on the AS/400.

NOTE: You can change all of the attributes of a field in the Field Properties dialog box but **only before you commit the file**. Once you commit it, the Progress/400 Data Dictionary restricts what attributes you can change.

CAUTION: Any DDS definitions for the field that are not supported by the DataServer will be lost.

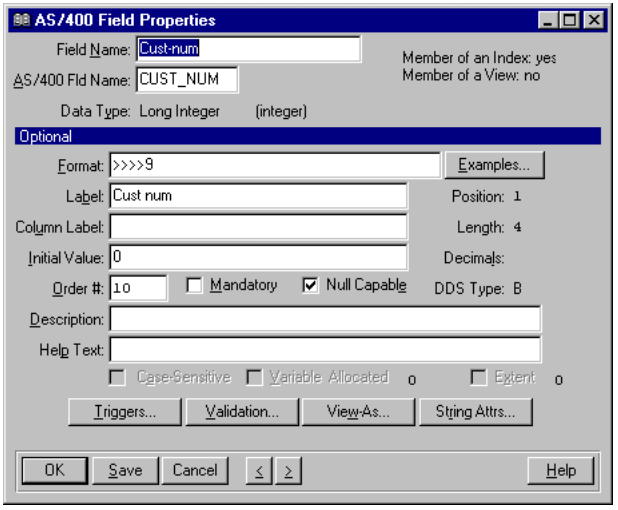
Follow these steps to modify a field:

1 ♦ From the Progress/400 Data Dictionary main window, choose the Fields mode button. The Fields list appears.

2 ♦ Select a table from the Tables list.

3 ♦ Select a field from the Fields list.

4 ♦ Choose the Field Properties button. The following dialog box appears:



5 ♦ Modify the fields to change the definitions for the DB2/400 file. Choose the Help button to access detailed information on all of the elements in the dialog box.

NOTE: You can override field-level validation expressions in your application by including the appropriate Progress 4GL statement.

6 ♦ Choose OK or Save. If you choose Save, use the navigation buttons to view more fields in the Field property panel.

You modified the data definitions in the server schema on the AS/400. Your server schema now contains definitions that are not reflected in the client schema holder until you synchronize it.

9.2.8 Deleting a Field

The Progress/400 Data Dictionary allows you to delete a field in the DB2/400 database file. You cannot delete fields that are part of an index without deleting the index first.

Follow these steps to delete a field:

1 ♦ Choose the Field mode button in the Progress/400 Data Dictionary main window.

2 ♦ Choose Modify schema.

3 ♦ Select the field that you want to delete.

4 ♦ Choose the Delete Field button. A message appears asking you to verify the deletion.

5 ♦ Choose Yes.

CAUTION: Whenever you delete a field, the data in the field is lost.

9.2.9 Adding an Index

The Progress/400 Data Dictionary allows you to add indexes (regular or word) to an existing DB2/400 file. Once an index is added, you can access it through Progress or AS/400 applications.

When you use the Progress/400 Data Dictionary to add regular (not word) indexes, the first index that you add becomes the Progress primary index. That index also becomes the physical file key on the AS/400, so no object is visible in the DataServer library. Note that if at least one index field is case sensitive, the entire index is case sensitive.

The Progress/400 Data Dictionary does not allow a word index to be a primary index. In addition, before you can create a word index, you must define a primary index.

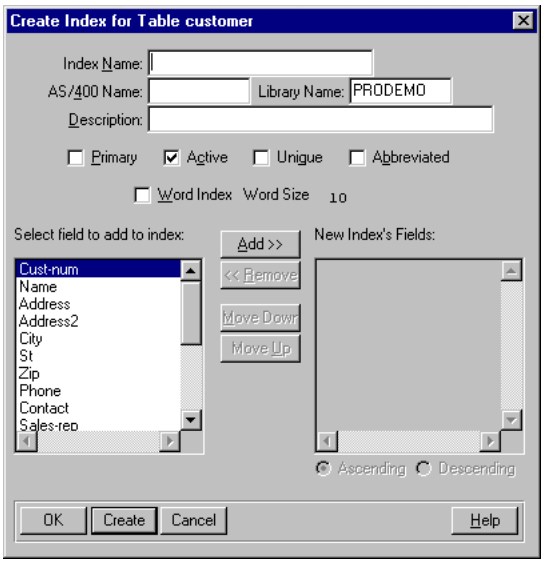
Follow these steps to add an index:

1 ♦ Choose the Index mode button in the Progress/400 Data Dictionary main window.

2 ♦ Choose Modify schema.

3 ♦ Select the table from the Tables list that will contain the new index.

4 ♦ Choose the Create Index button. The following dialog box appears:



5 ♦ Enter a unique name in the Index Name field.

6 ♦ Press **TAB** and a name for the AS/400 logical file is supplied for you based on the index name. For example, for an index named ncust-num, the AS/400 logical file name is NCUST_NUM.

7 ♦ Activate the Primary, Unique, Abbreviated, or Word Index check box as required.

8 ♦ If you selected the Word Index check box, enter the word size in the Word Size field. You must specify a value from 10 to 128. This value specifies the length of the longest word that is entered into the text field on which the index is built. A larger value results in a larger index.

NOTE: You can change the word size in the Index Properties dialog box but **only before you commit the word index**. Once you commit it, you can change the word size only by deleting and re-creating the word index.

9 ♦ Select one or more fields to include in the index, then choose Add.

10 ♦ Choose OK, or choose Create if you want to create another index.

9.2.10 Modifying an Index

The Progress/400 Data Dictionary allows you to modify DB2/400 indexes (logical files). The DataServer takes the definitions you provide and adds them to the logical file on the AS/400.

Follow these steps to modify an index:

1 ♦ Choose the Index mode button in the Progress/400 Data Dictionary main window.

2 ♦ Choose Modify schema.

3 ♦ Select the table and index that you want to modify.

4 ♦ Choose the Index Properties button. The following dialog box appears:



5 ♦ You can change the names, move to a different library, or change the descriptions. Choose the Help button to access detailed information on all of the elements in the dialog box. If your font is too large to fit all the characters in a field, scroll the field with the arrow key.

If you flag the index as inactive, a message box appears giving you the option of deleting the logical file on the AS/400.

NOTE: You can change the word size in the Index Properties dialog box **only before you commit the word index**. Once you commit it, you can change the word size only by deleting and re-creating the word index.

6 ♦ Choose OK or choose Save. Choose one of the navigation buttons to view more Index property sheets.

9.2.11 Deleting an Index

The Progress/400 Data Dictionary allows you to delete an index in the server schema. It also gives you the option of deleting the associated DB2/400 logical file. If you choose to delete the index only from the server schema, you can still access the associated logical file through non-Progress AS/400 applications. However, if you choose to delete the logical file, no application can access it.

Follow these steps to delete an index:

1 ♦ Choose the Index mode button in the Progress/400 Data Dictionary main window.

2 ♦ Choose Modify schema.

3 ♦ Select the index that you want to delete.

4 ♦ Choose the Delete Index button. A message appears asking you to verify the deletion.

5 ♦ Choose Yes. A message appears asking whether you want to delete the DB2/400 logical file.

6 ♦ Choose Yes to delete the logical file, or choose No to delete the index from the server schema only and preserve the logical file.

9.2.12 Adding a Sequence

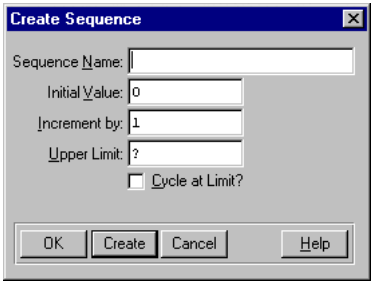
The Progress/400 Data Dictionary allows you to add sequences to the server schema. You can then access those sequences through Progress applications.

Follow these steps to add a sequence:

1 ♦ Choose the Sequence mode button in the Progress/400 Data Dictionary main window.

2 ♦ Choose Modify schema.

3 ♦ Choose the Create Sequence button. The following dialog box appears:



4 ♦ Enter a unique name in the Sequence Name field.

5 ♦ Enter values in the remaining fields.

6 ♦ Activate the Cycle at Limit check box, if required.

7 ♦ Choose OK, or choose Create if you want to create another sequence.

9.2.13 Modifying a Sequence

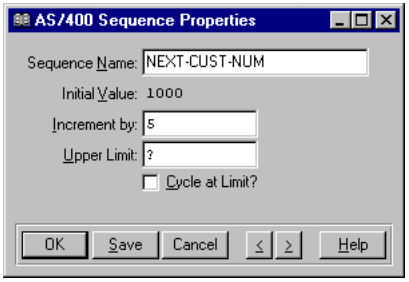
The Progress/400 Data Dictionary allows you to modify sequences. Follow these steps to modify a sequence:

1 ♦ Choose the Sequence mode button in the Progress/400 Data Dictionary main window.

2 ♦ Choose Modify schema.

3 ♦ Select the sequence to modify.

4 ♦ Choose the Sequence Properties button. The following dialog box appears:



5 ♦ Modify the fields to change the definitions for the sequence. Choose the Help button to access detailed information on all of the elements in the dialog box.

6 ♦ Choose OK or choose Save. Then choose one of the navigation buttons to view more Sequence Property sheets.

9.2.14 Deleting a Sequence

The Progress/400 Data Dictionary allows you to delete a sequence in the server schema.

Follow these steps to delete a sequence:

1 ♦ Choose the Sequence mode button in the Progress/400 Data Dictionary main window.

2 ♦ Choose Modify schema.

3 ♦ Select the sequence that you want to delete.

4 ♦ Choose the Delete Sequence button. A message appears asking you to verify the deletion.

5 ♦ Choose Yes.

9.2.15 Adding a Procedure

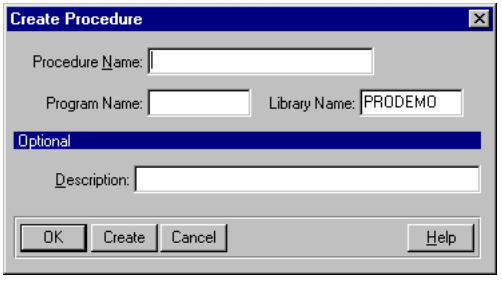
The Progress/400 Data Dictionary allows you to add a stored procedure definition to the Dictionary Library. The Progress/400 Data Dictionary stores the definition in the P__FILE file, so that you can evoke the stored procedure through Progress applications.

Follow these steps to add a procedure:

1 ♦ Choose the Procedure mode button in the Progress/400 Data Dictionary main window.

2 ♦ Choose Modify Schema.

3 ♦ Choose the button. The following dialog box appears:



4 ♦ Enter a unique name in the Procedure Name field. This is the name by which Progress recognizes this object, so it should follow Progress naming conventions.

5 ♦ Enter both the Program Name of the object on the AS/400 and the Library Name where the program resides. The Program Name name must be unique within the server schema and the name must differ from any existing physical file in the library.

6 ♦ Enter a description in the Description field. Choose the Help button to access detailed information on all of the elements in the dialog box.

7 ♦ Choose OK to advance to the Define Parameter screen or choose Create to create the record and begin defining another stored procedure.

If you want to create parameters for this stored procedure, see the "Adding a Parameter" section.

8 ♦ Create parameters for this stored procedure if applicable.

You modified the data definitions in the server schema on the AS/400. Your server schema now contains an object that is not reflected in the client schema holder until you synchronize it.

9.2.16 Modifying a Procedure

The Progress/400 Data Dictionary allows you to modify DB2/400 stored procedure definitions.

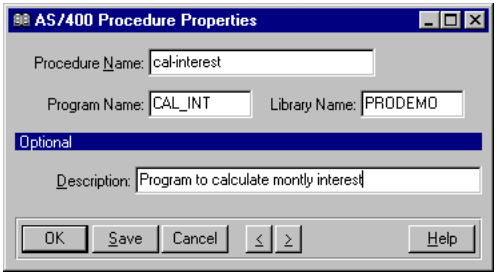
Follow these steps to modify a stored procedure definition:

1 ♦ Choose the Procedure mode button in the Progress/400 Data Dictionary main window.

2 ♦ Choose Modify Schema.

3 ♦ Select a stored procedure from the stored procedure list.

4 ♦ Choose the Procedure Properties button. The following dialog box appears:



5 ♦ Modify the fields to change the definition for the DB2/400 program. Choose the Help button to access detailed information on all of the elements in the dialog box.

6 ♦ Choose OK to return to the Progress/400 Data Dictionary.

You modified the data definitions in the server schema on the AS/400. Your server schema now contains an object that is not reflected in the client schema holder until you synchronize it.

9.2.17 Deleting a Procedure

The Progress/400 Data Dictionary allows you to delete a stored procedure definition from the server schema. This process does not delete the program from the AS/400.

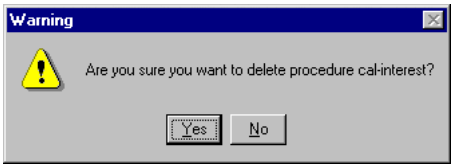
Follow these steps to delete a stored procedure definition from the server schema:

1 ♦ Choose the Procedure mode button in the Progress/400 Data Dictionary main window.

2 ♦ Choose Modify schema.

3 ♦ Select the stored procedure from the stored procedure list.

4 ♦ Choose the Delete Procedure Button. The following message appears:



5 ♦ Choose Yes.

You have modified the data definitions in the server schema on the AS/400. Your server schema now has changes that are not reflected in the client schema holder until you synchronize it.

9.2.18 Adding a Parameter

The Progress/400 Data Dictionary allows you to add parameters to existing DB2/400 stored procedures. You can then use these parameters when running the stored procedure in your Progress application.

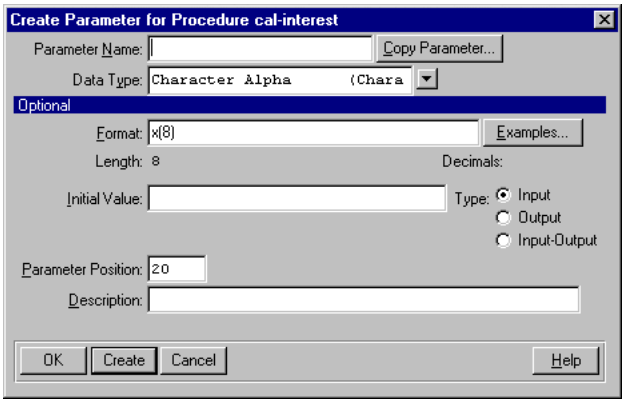
Follow these steps to add a parameter:

1 ♦ Choose the Parameter mode button in the Progress/400 Data Dictionary main window.

2 ♦ Choose Modify schema.

3 ♦ Select the stored procedure from the procedure list that will contain the new parameter.

4 ♦ Choose the Create Parameter button. The following dialog box appears:



5 ♦ Enter a unique name in the Parameter Name field. This is the name by which Progress recognizes this object, so it should follow Progress naming conventions.

6 ♦ Choose a data type.

Table 9–3 lists the supported data types and their Progress equivalents.

Table 9–3: DB2/400 Data Types with Progress Equivalents

DB2/400	Progress
Character Alpha	Character
Zoned Numeric	Decimal
Packed Decimal	Decimal
Pckd (even decimal)	Decimal
Long Integer	Integer
Short Integer	Integer
Character	Logical

A default format is supplied based on the data type. You can change the default format. The format or datatype determines the size of the parameter.

7 ♦ Select the type of parameter being defined. There are three types: input to the stored procedure, output from the stored procedure, or input-output from the stored procedure.

8 ♦ Enter any other information that you want to add to the new parameter definition. Choose the Help button to access detailed information on all of the elements in the dialog box.

9 ♦ Choose OK to return to the Progress/400 Data Dictionary main window or choose Create to create the record and define another one. Up to 32 parameters can be defined for each stored procedure.

You modified the data definitions in the server schema on the AS/400. Your server schema now contains definitions that are not reflected in the schema holder until you synchronize it.

9.2.19 Modifying a Parameter

The Progress/400 Data Dictionary allows you to modify the parameters to stored procedure definitions.

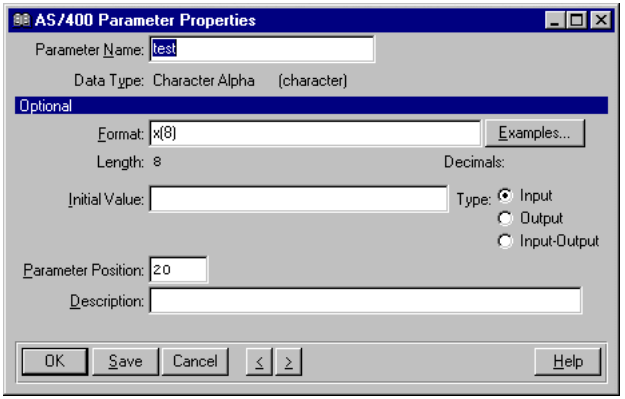
Follow these steps to modify a parameter:

1 ♦ From the Progress/400 Data Dictionary main window choose the Parameter mode button.

2 ♦ Select the stored procedure from the procedure list.

3 ♦ Select a parameter from the parameter list.

4 ♦ Choose the Parameter Properties button. The following dialog box appears:



5 ♦ Modify the parameter to change the definitions. Choose the Help button to access detailed information on all of the elements in the dialog box.

6 ♦ Choose OK or Save. If you choose Save, use the navigation buttons to view more parameters in the Parameter property panel.

You modified the data definitions in the server schema on the AS/400. Your server schema now contains definitions that are not reflected in the client schema holder until you synchronize it.

9.2.20 Deleting a Parameter

The Progress/400 Data Dictionary allows you to delete a parameter definition.

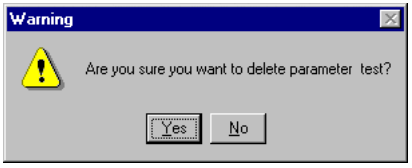
Follow these steps to delete a parameter:

1 ♦ Choose the Parameter mode button in the Progress/400 Data Dictionary main window.

2 ♦ Choose Modify schema.

3 ♦ Select the parameter you want to delete.

4 ♦ Choose the Delete Parameter button. The following message appears:



5 ♦ Choose Yes.

You have modified the data definitions in the server schema on the AS/400. Your server schema now has changes that are not reflected in the client schema holder until you synchronize it.

9.3 Progress/400 Data Dictionary Administration Utilities

The administrative utilities described in the following sections are incorporated in the Progress/400 Data Dictionary. They are available from the Admin menu. When you use the Progress/400 Data Dictionary in read-only mode, you have access only to:

- Dump Data & Definitions
- Dump Table Contents
- Dump Sequences Definitions

- Dump Sequences Current Values
- Load Table Contents
- Reconstruct Bad Load Records
- Progress/400 Synchronize (client)

All other utilities require that you use the Progress/400 Data Dictionary in modify-schema mode.

To alternate between modify-schema mode and read-only mode, activate the appropriate radio button in the Progress/400 Data Dictionary main window. Be sure to commit changes that you make when you move from modify-schema mode to read-only mode in the Progress/400 Data Dictionary.

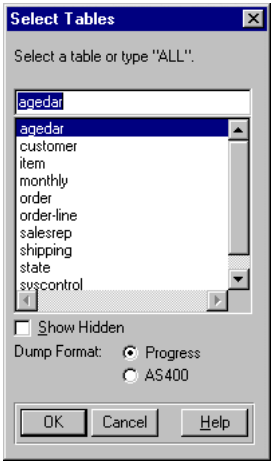
9.3.1 Dumping DB2/400 Data Definitions

You use the Progress/400 Dump Data Definitions utility to dump definitions from DB2/400 database files. The utility dumps information stored in the server schema on the AS/400. You can choose whether the resulting data definitions file (.df) has a Progress or an AS/400 format.

Follow these steps to dump DB2/400 data definitions:

1 ♦ Access the Progress/400 Data Dictionary.

2 ♦ Choose Admin→Dump Data & Definitions→Data Definitions (`.df`). The following dialog box appears:



3 ♦ Select the tables from which you want to dump data definitions.

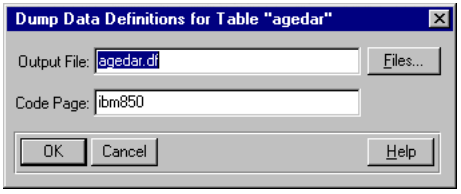
4 ♦ Activate the Progress or AS400 radio button to select a Progress or AS/400 format for the .df file:

- Progress format - Choose this format if you want to preserve only Progress information. This creates a standard .df file that you can then load into a Progress database or a DB2/400 database.
- AS/400 format - Choose this format if you want to preserve non-Progress information, such as DDS data type, field storage length, or AS/400 name. You cannot load a .df file with an AS/400 format into a standard Progress database. You cannot use the standard Progress load utility to load an AS/400 formatted .df file.

CAUTION: If dumping in AS/400 format, you can only load the file with the Progress/400 Data Dictionary.

5 ♦ Choose OK.

The following dialog box appears:



6 ♦ Enter a name for the data definitions (.df) file or accept the default name.

7 ♦ Choose OK.

When virtual files are dumped and reloaded into another server schema, they become physical files, not virtual files.

The server schema might contain only a subset of the DDS data definitions for your original DB2/400 files. The resulting .df file does not contain information regarding unsupported DDS definitions.

9.3.2 Dumping Data

Follow these steps to dump data from DB2/400 database files:

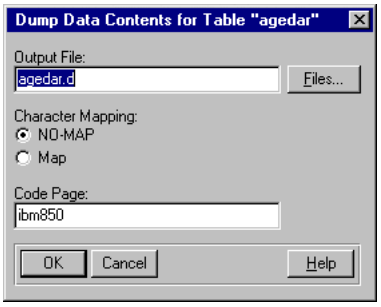
1 ♦ Access the Progress/400 Data Dictionary.

2 ♦ Make sure that the client schema holder is synchronized.

3 ♦ Choose Admin→ Dump Data & Definitions→ Table Contents (.d).

4 ♦ Select the table or tables from which you want to dump data.

5 ♦ Enter a name for the data (.d) file. By default the filename is the database name with the .d extension. The following dialog box appears:



6 ♦ Choose OK.

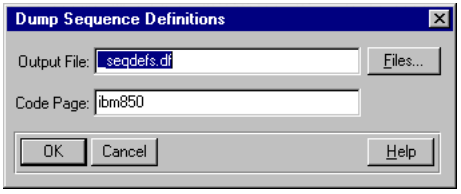
You can load the resulting data file into DB2/400 database files using the load utility in the Progress/400 Data Dictionary. Additionally, you can load data into the DB2/400 files through the standard Progress load utilities. Make sure that the client schema holder has been synchronized.

9.3.3 Dumping Sequence Definitions

Follow these steps to dump definitions of sequences from DB2/400 database files:

1 ♦ Access the Progress/400 Data Dictionary.

2 ♦ Choose Admin→ Dump Data & Definitions→ Sequences Definitions (.df). The following dialog box appears:



3 ♦ Enter a name for the definition (.df) file. The filename is `_seqdefs.df` by default.

4 ♦ Choose OK.

You can load the resulting data definitions file into DB2/400 database files using the load utility in the Progress/400 Data Dictionary.

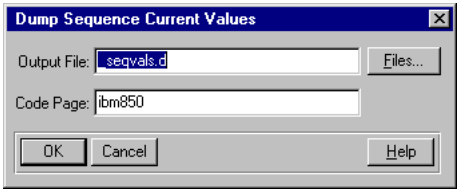
9.3.4 Dumping Sequence Current Values

Follow these steps to dump the current values of sequences from DB2/400 database files:

1 ♦ Access the Progress/400 Data Dictionary.

2 ♦ Make sure that the client schema holder is synchronized.

3 ♦ Choose Admin→ Dump Data & Definitions→ Sequences Current Values (.d). The following dialog box appears:



4 ♦ Enter a name for the data (.d) file. By default the filename is `_seqvals.d`

5 ♦ Choose OK.

You can load the resulting data file into DB2/400 database files using the load utility in the Progress/400 Data Dictionary.

9.3.5 Creating Incremental .df Files

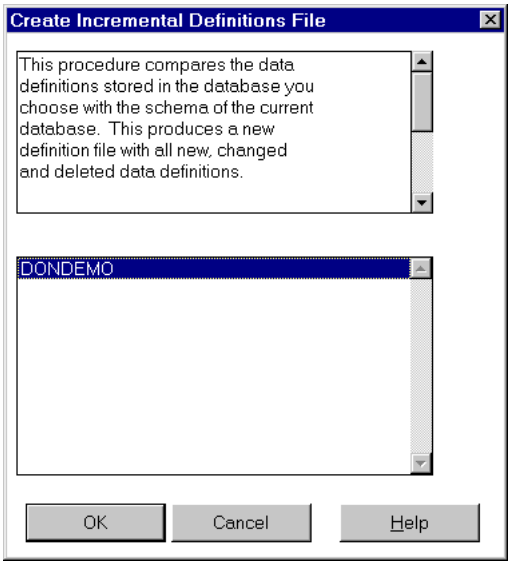
You use the Progress/400 incremental file utility to compare two Progress/400 server schemas and create a .df file that contains any differences. You can then use the incremental .df file to upgrade from an existing DB2/400 database to the current DB2/400 database.

NOTE: You must have at least two DB2/400 Databases connected to create an incremental .df file. Your working database should be the database with the newest version of the database schema. Progress/400 creates the incremental .df file in a Progress format only, which might not preserve DB2/400 data types. See Table 9–4 for Progress-to-DB2/400 data type mapping.

Follow these steps to create an incremental file:

1 ♦ Access the Progress/400 Data Dictionary.

2 ♦ Choose Admin→ Dump Data & Definitions→ Create Incremental .df File. The following window appears:



The Progress/400 Data Dictionary lists all connected DB2/400 databases except the working database.

3 ♦ If you have more than two DB2/400 databases connected, select the database that you want to update to be a duplicate of your working database.

4 ♦ When prompted, enter the name of the file to which the utility will write the differences. The default is DELTA.DF.

5 ♦ Choose OK to perform the comparison. The file, field, and index names are displayed during the comparison.

Once the comparison is complete and the incremental .df file has been created, you can load the file using the Progress/400 Data Dictionary to apply the schema changes to an existing database. After you load the file, you must synchronize the schema holder. See the"Loading Data Definitions" and "Synchronizing the Client" sections, respectively, for details.

9.3.6 Loading Data Definitions

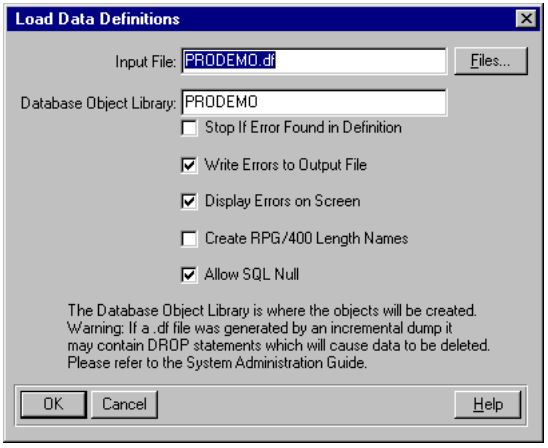
You use the Progress/400 load utility to move your Progress database to the AS/400 after you create the server schema. Note that when you load Progress-formatted or AS/400-formatted data definitions into DB2/400 database files, you lose any DDS definitions in the DB2/400 database files that the DataServer does not support. (See the *Progress/400 Release Notes* for a list of supported DDS keywords.)

Follow these steps to load data definitions into the server schema:

1 ♦ Access the Progress/400 Data Dictionary and choose Modify schema.

2 ♦ Choose Admin→ Load Data & Definitions→ Data Definitions (.df).

The following dialog box appears:



3 ♦ Enter the name of the data definitions (.df) file that you want to load.

You cannot load a .df file of a type other than AS/400 or Progress (for example, ORACLE).

You can load a standard Progress .df file (including an incremental .df file) or an AS/400-formatted .df file. If you load a standard Progress .df file, the utility adds the information required by the AS/400, such as field storage length and DDS data type.

CAUTION: Use the Progress/400 Data Dictionary to load a .df file. Additionally, if you load an AS/400-formatted .df file that you have modified by hand, unpredictable results can occur.

Table 9–4 describes how the utility maps Progress data types to DDS and DB2/400 data types. See the "Data Types" section in Chapter 2, "Common Product Information," for more information.

Table 9–4: Progress-to-DB2/400 Data Type Mapping

Progress	AS/400	DDS
CHARACTER	String	A
DATE	Date *ISO	L
DECIMAL	Packed decimal	P
INTEGER	Long integer-two-byte or four-byte, depending on the size of the display format	B
LOGICAL	Character	A

4 ♦ Activate a toggle box to select how the load utility handles errors and warnings. Even if you do not choose to display errors on screen, the utility displays serious errors; that is, errors that prevent the load from continuing or backing everything out.

5 ♦ If you want all file and field names to be RPG compliant, activate the Create RPG/400 Length Names toggle box.

6 ♦ If you want all fields to be SQL Null Capable, accept the default value, otherwise deactivate the Allow SQL Null toggle box.

7 ♦ Choose OK.

If the .df file contains an index the length of whose combined fields is greater than or equal to 200, the load utility does not create the index. However, the load utility continues to load the valid definitions and displays messages about definitions that it could not load.

Loading a definition file that contains word indexes creates an index with a default word size of 30.

NOTE: You can change the word size in the Index Properties dialog box, but **only before you commit the word index**. Once you commit it, you can change the word size only by deleting and re-creating the word index. Similarly, you can change the attributes of a field in the Field Properties dialog box, but **only before you commit the file**. Once you commit it, the Progress/400 Data Dictionary restricts what attributes you can change.

If the .df file contains virtual file definitions, the load utility creates physical files. For example, when you dump definitions for a logical join named CUSTJ1 and load it, the DataServer creates a physical file CUSTJ1 and a logical file CUSTJ1__1 for the primary index. See the "Virtual Tables" section in Chapter 2, "Common Product Information," for more information.

9.3.7 Loading Data

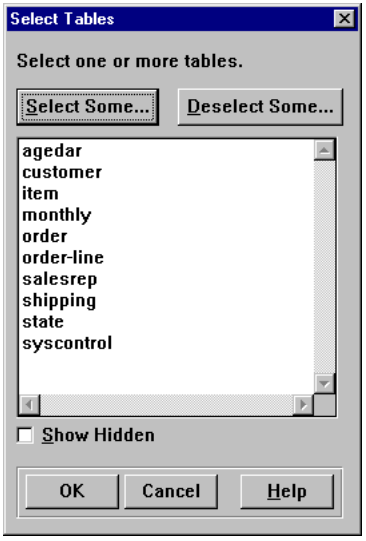
Follow these steps to load data into DB2/400 database files:

1 ♦ Access the Progress/400 Data Dictionary.

2 ♦ Make sure that the client schema holder is synchronized.

3 ♦ Choose Load Data & Definitions→ Table Contents (.d).

4 ♦ The following dialog box appears:



5 ♦ Enter the name of the data (.d) file that you want to load and choose OK.

You can also use the standard Progress load utilities in the Data Administration tool to load data into DB2/400 database files.

9.3.8 Loading Sequence Current Values

Follow these steps to load definitions for sequences into DB2/400 database files:

1 ♦ Access the Progress/400 Data Dictionary and choose Modify Schema.

2 ♦ Choose Admin→ Load Data & Definitions→ Sequences Current Values (.d).

3 ♦ Enter the name of the data (.d) file that you want to load.

4 ♦ Choose OK.

9.3.9 Reconstructing Bad Load Records

Use the Reconstruct Bad Load Records utility to load error (.e) files that were generated by standard Progress or Progress/400 load utilities. This utility is the standard utility in the Progress Data Administration tool. The Progress/400 Data Dictionary Admin menu provides access to it for your convenience.

9.3.10 Synchronizing the Client

You use the Progress/400 Dictionary Synchronize utility to synchronize either the client schema holder and the Native 4GL Client Schema Image or just the client schema holder, so that it matches the information in the server schema on the AS/400. You can choose to perform either a full or a selective synchronization:

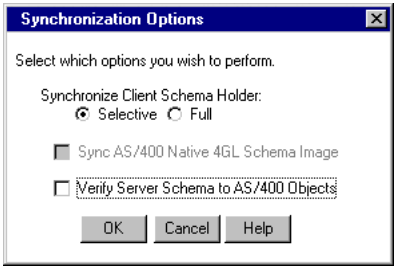
- A full synchronization synchronizes all files.
- A selective synchronization synchronizes only those files that you specify.

NOTE: When performing a selective synchronization, this utility does not check for or remove objects from the schema holder that are no longer present in the server schema. It also does not process changes to sequences. You must perform a full synchronization to include this activity.

Follow these steps to synchronize the client:

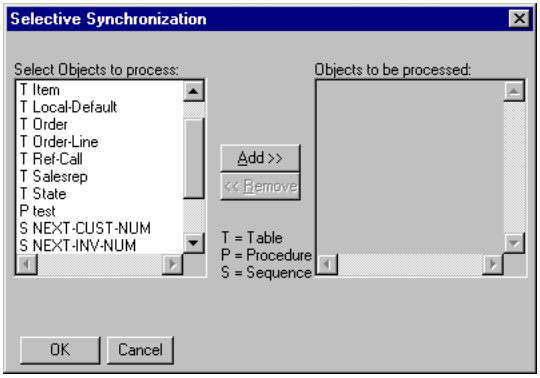
1 ♦ Access the Progress/400 Data Dictionary.

2 ♦ Choose Admin→ Synchronize. The following dialog box appears:



3 ♦ Choose the Selective or Full radio button to perform a selective or a full synchronization.

If you choose selective synchronization, the following dialog box appears:



This window displays the files that you can select for synchronization.

4 ♦ If you chose the Selective button, choose the files to synchronize:

- To select a file, use the up and down arrow keys in the Select Objects to process: pane to highlight a filename, then press the **RETURN** key or the Add >> button to select it. The file name moves to the Objects to be processed: pane. Repeat this process for each object to be synchronized.
- To remove a selected object, use the up and down arrow keys in the Objects to be processed: pane to highlight the object name, then press the **RETURN** key or the << Remove button to remove it. The object name moves to the Select Objects to process: pane. Repeat the selection process for each object to be removed.

After your selection list is complete, choose OK.

5 ♦ Choose to synchronize your client schema holder and/or your Native 4GL schema image.

6 ♦ Choose Verify Server Schema to verify the physical file objects against the information in the server schema. If the utility finds any differences, it displays a report.

7 ♦ Choose OK.

9.3.11 Freezing/Unfreezing a Table

Follow these steps to freeze or unfreeze a table:

1 ♦ Access the Progress/400 Data Dictionary and choose Modify schema.

2 ♦ Select the table you want to freeze or unfreeze.

3 ♦ Choose Admin→ Freeze/Unfreeze. The following dialog box appears:



4 ♦ Select the Frozen toggle box to freeze or unfreeze the table.

5 ♦ Choose OK to return to the Data Dictionary window.

10

AS/400 Utilities

This chapter describes the utilities that you use to create and maintain your Progress/400 environment on the AS/400. This chapter groups the utilities according to the subject areas in the following list:

- Running Server Utilities
- Progress/400 AppServer Commands
- Progress/400 Dictionary & Utility Commands
- Progress/400 Word Index Support Commands
- Progress/400 TCP/IP Network Commands
- Progress/400 Native 4GL Client Commands

10.1 Progress/400 Utilities Grouped by Subject Area

This section lists all of the subject areas for the Progress/400 utilities, as well as the names of the utilities contained in each.

Progress/400 AppServer Commands

The following list contains the names of the utilities you use to create and maintain the Progress/400 AppServer. For a detailed description of each utility, see the "Progress/400 AppServer Commands" section in this chapter:

- **ENDADMSVR** — End Progress/400 AdminServer Utility
- **SETPROENV** — Set Progress/400 Environment Utility
- **STRADMSVR** — Start Progress/400 AdminServer Utility

Progress/400 Dictionary Library & Utility Commands

The following list contains the names of the utilities you use to create and maintain a Progress/400 dictionary library. For a detailed description of each utility, see the "Progress/400 Dictionary Library & Utility Commands" section in this chapter:

- **CHGPRODCT** — Change Progress/400 Dictionary Utility
- **CHGPRODFT** — Change Progress/400 Defaults Utility
- **CRTPROLKT** — Create Progress/400 Lock Table Utility
- **CRTSCHIMG** — Create Schema Image Utility
- **CVTPRODCT** — Convert Progress/400 Dictionary Utility
- **CVTSRVSCH** — Convert Server Schema Utility
- **DUPPRODB** — Duplicate Progress/400 Database Utility
- **DMPPRODTA** — Dump Data Utility
- **LODPRODTA** — Load Data Utility
- **RMVSCHE** — Remove Server Schema Entry Utility
- **SYNSCHIMG** — Synchronize Schema Image Utility

Progress/400 Word Index Support Commands

The following list contains the names of the utilities you use to create and maintain Progress/400 word indexes. For a detailed description of each utility, see the "Progress/400 Word Index Support Commands" section in this chapter:

- **ENDWISPRC** — End Word Index Support Processor Utility
- **GENWRDIDX** — Generate Word Index Utility
- **RPLDCTWBT** — Replace Dictionary Word Break Table Utility
- **STRWISPRC** — Start Word Index Support Processor Utility
- **UPDPROWBT** — Update Progress Word Break Table Utility
- **UPDWISTRG** — Update Word Index Support Triggers Utility

Progress/400 TCP/IP Network Commands

The following list contains the names of the utilities you use to create and maintain Progress/400 networking. For a detailed description of each utility, see the "Progress/400 TCP/IP Network Commands" section in this chapter:

- **ENDPRONET** — End Progress/400 Network Utility
- **MNGPRONET** — Manage Progress/400 Networking Utility
- **STRPRONET** — Start Progress/400 Networking Utility
- **WRKPROJOB** — Work With Progress Jobs Utility

Progress/400 Native 4GL Client Commands

The following list contains the name of the utility you use to start the Native 4GL Client. For a detailed description of this utility, see the "Progress/400 Native 4GL Client Commands" section in this chapter:

- **STRPROCLI** — Start Native 4GL Compiler Utility

10.2 Running Server Utilities

On the AS/400, you can run most Progress/400 server utilities either from a menu or from the command line. A few utilities can be run only from the command line.

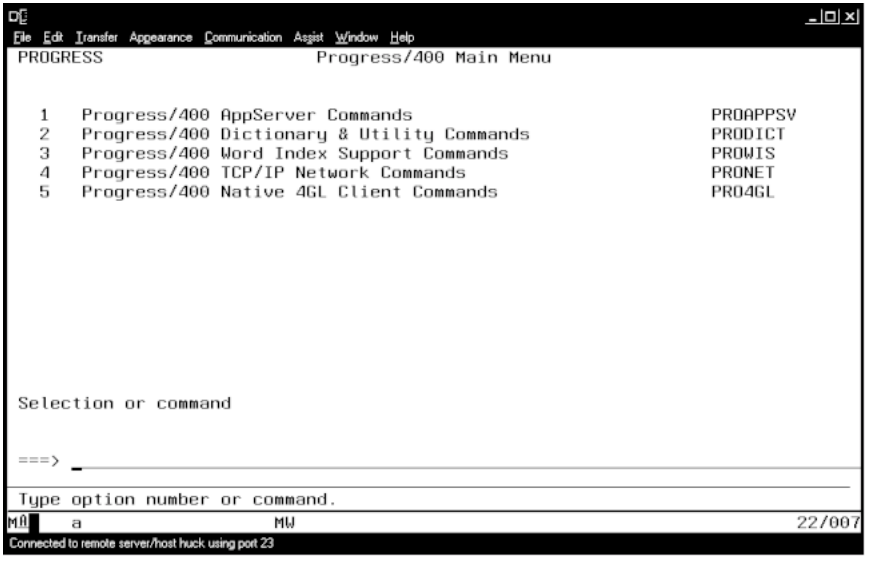
10.2.1 Running Server Utilities from the Menu

Follow these steps to run a utility from the **menu**:

1 ♦ Type the following command on the command line, then press **F4**:

```
GO PROGRESS
```

The following menu appears:



Note that this menu is divided into five sections by utility type: AppServer, Dictionary, Word Index, TCP/IP Network, and Native 4GL Client. If a Progress/400 command is not listed within one of the groups in this menu, you must run it from the command line.

2 ♦ Type the number of the utility to run after the arrow, then press **RETURN**.

10.2.2 Running Server Utilities from the Command Line

To run a utility from the command line, follow these steps:

1 ♦ Type the command on the command line, then press **F4**.

2 ♦ Press **F10** to see all possible parameters for a given command.

3 ♦ Accept the default values or type the appropriate value.

You can also run the utility by entering the command, the keywords for parameters, and the appropriate values at the command line.

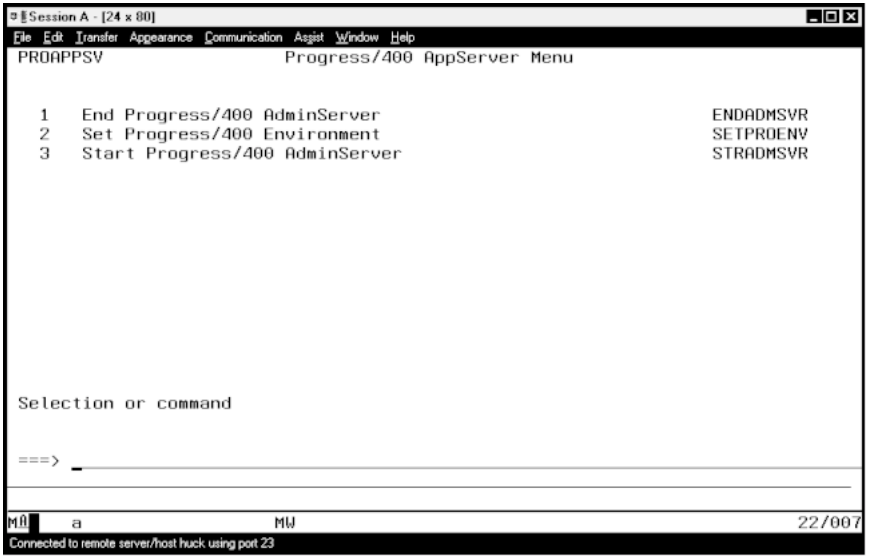
10.2.3 Accessing Online Help

Progress/400 provides online help for each of the Progress/400 server utilities described in this chapter. There are three ways to access online help:

- Enter the Progress/400 utility name on the command line and press the **F1** key.
- Use OS/400 command prompting (press **F4**).
- Position the cursor on the parameter for which you need the help description and press **F1**.

10.3 Progress/400 AppServer Commands

This section explains when to use the Progress/400 AppServer commands and documents the field definitions for each. When you select Progress/400 AppServer Commands from the Progress/400 Main Menu screen, the following menu appears:



Type the number of the utility to run after the arrow, then press **RETURN**.

10.3.1 End Progress/400 AdminServer (ENDADMSVR)

Use the ENDADMSVR utility to stop the Progress/400 AdminServer. ENDADMSVR requires a username and password to prevent unauthorized users from randomly shutting down the AdminServer. This utility submits the ENDADMSVR job to the AdminServer subsystem. With the successful completion of the ENDADMSVR job, the AdminServer shuts down and the subsystem ends.

Table 10–1 shows the ENDADMSVR utility parameters and values.

Table 10–1: ENDADMSVR Parameters

Parameter	Value
USER	Enter a valid AS/400 User profile
PASSWORD	Enter the password for the User profile

10.3.2 Set Progress/400 Environment (SETPROENV)

The SETPROENV command sets up environment variables used by the AppServer. The SETPROENV command accepts the parameters shown in Table 10–2:

Table 10–2: SETPROENV Parameter

Parameter	Value
PROPATH	Enter directories separated by commas (,). The directories are prepended to the default Progress/400 propath.

The SETPROENV command affects the environment variables shown in Table 10–3.

Table 10–3: Environment Variables Affected by SETPROENV

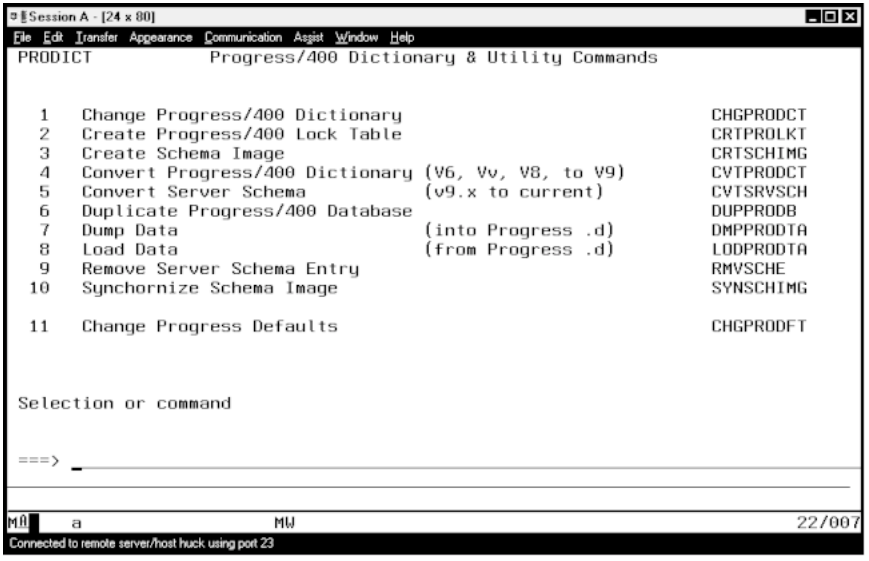
Parameter	Value
ADMSVRSBS	ADMSVRSBS names the Progress/400 AdminServer subsystem. You cannot change this value.
CLASSPATH	The default Java CLASSPATH for the Progress/400 product. This value can be changed, but nothing should be removed.
DLC	This value is taken from the Progress/400 installation. It is the directory where your DBA installed Progress/400. Typically the directory name follows the convention `/PROGRESS/V91A`
JREHOME	The Progress/400 product defines this variable, which matches the DLC variable.
PROMSGS	The location of the PROMSGS file in the root file system, which Progress always sets to `$DLC/promsgs`.
PROPATH	This is the default PROPATH for Progress/400. Enter directory names separated by commas (,). Progress/400 prepends your list to the PROPATH.
QIBM_MULTI_THREADED	Causes the OS/400 QSHELL interpreter to start with multi-threaded capabilities. Progress/400 sets this value to Y. This value must not change.
QIBM_CHILD_JOB_SNDINQMSG	Progress/400 sets this value to 0. This value must not change.
WRKDIR	Specified when you install the Progress/400 products.

10.3.3 Start Progress/400 AdminServer (STRADMSVR)

The STRADMSVR command starts the Progress/400 AdminServer.

10.4 Progress/400 Dictionary Library & Utility Commands

This section explains when to use the Progress/400 dictionary library and utility commands and documents the field definitions for each. When you select Progress/400 Dictionary Library and Utility Commands from the Progress/400 Main Menu screen, the following menu appears:



Type the number of the utility to run after the arrow, then press **RETURN**.

10.4.1 Change Progress/400 Dictionary Utility (CHGPRODCT)

Use the CHGPRODCT utility to accomplish the following tasks:

- Migrate DB2/400 database files to the Progress/400 environment.
- Add DB2/400 database files to an existing Progress/400 server schema.
- Update a Progress/400 server schema to reflect changes made to supporting DB2/400 database files through DDS or SQL.
- Synchronize your Native 4GL Client.

Note that you cannot use CHGPRODCT to delete files from the Progress/400 server schema. You must use the RMVSCHE utility instead. See the "Remove Server Schema Entry (RMVSCHE)" section for details.

Table 10–4 describes the CHGPRODCT parameters.

Table 10–4: CHGPRODCT Parameters *(1 of 2)*

Parameter	Keyword	Value
Progress/400 Dictionary Name	PRODCT	Enter the name of the library that contains your server schema, or *CURLIB to specify the current library. If you are creating a Progress/400 environment for the first time, enter here the same name that you entered for the New Progress/400 Database Name in the DUPPRODB utility.
From File(s)	FRMFILLST	Specify the files that you want to access through the DataServer. Accept the default, *ALL, if you do not want to specify files. Press + to enter additional files, or specify other values as follows: - If you enter *YES at the include Database Relations parameter, you need to enter only the physical filenames in the From File list and not the names of the database relations. - If you enter *NO or *NONE, do not include any database relations over the physical file in the server schema. - If you enter *PFLIB, only the database relations residing in the library within the physical file from which they are based are included in the server schema.
From Library(s)		Enter the name of the library where the DB2/400 database files reside. If the objects reside in more than one library, enter *LIBL to specify the library list.
Include Database Relations		Enter *YES to preserve the logical file dependencies (including indexes) that exist for the DB2/400 database files that you selected. You need to enter only the physical filenames in the From File list and not the names of the database relations.

Table 10–4: CHGPRODCT Parameters *(2 of 2)*

Parameter	Keyword	Value
Progress Attributes Option	PROATR	Enter *OVRWRT to overwrite any Progress-specific attributes defined previously for this file from the client in the Progress/400 Data Dictionary. Enter *CMPSAVE to preserve attributes that are maintainable from the client if they already exist in the Progress/400 Dictionary. Attributes that can be defined from either the Progress/400 Dictionary on the client or the DDS on the server are not preserved. In these cases, the values in the file attributes are compared with those that exist in the Progress/400 Dictionary. If the value changes, the new DDS file attributes replace the existing one. If the existing Progress attribute does not change, then it is retained in the Progress/400 Dictionary definition. Enter *SAVE to preserve Progress attributes that are maintainable from the client if they already exist in the Progress/400 Dictionary for a specific file.
Force Server-based Change	FRCCHG	When using this utility to update the Progress/400 server schema, enter *YES to overwrite any Progress-specific attributes that you defined from the client ADE. Enter *NO to preserve the Progress attributes if a selected file was last updated from the Progress client. The utility does not update these files and returns an error message indicating that it did not update them.
Error Tolerance Threshold	ERRLIMIT	Enter *NOMAX to specify that the utility continue when an error occurs. Enter a value (0-32,000) to indicate an acceptable level of error.
Synchronize Schema Image	SYNSCHIMG	Enter *YES to synchronize the schema image for your Native 4GL Client. The default is *NO.

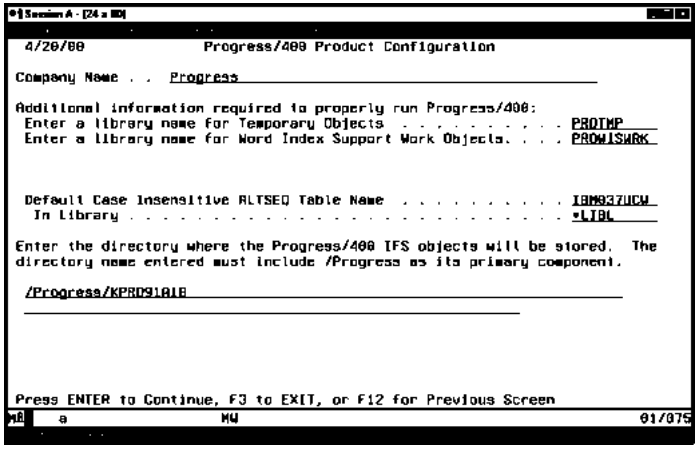
10.4.2 Change Progress/400 Defaults Utility (CHGPRODFT)

Use the CHGPRODFT utility to change the values in the user space that contains Progress/400 installation values.

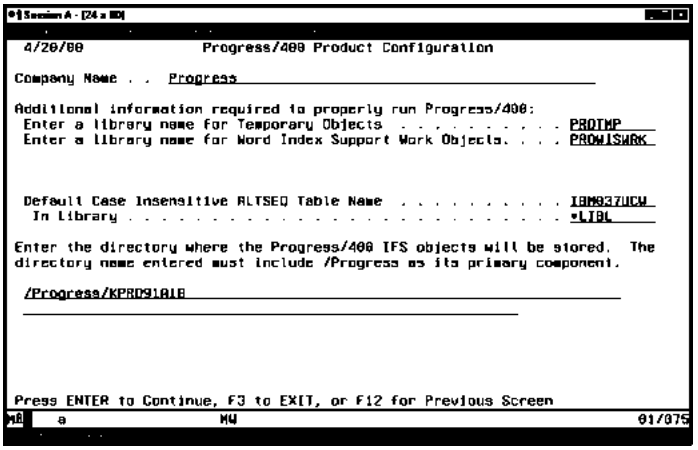
When you initially install Progress/400, the installation utility places many of the installation values in a user space in the Progress library. These values include, for example, the name of the default case-insensitive table and the name of the IFS directory where Progress/400 stores IFS objects.

If you want to rename the installed library or copy its contents to another library, you must reinitialize the values in this user space. To do this, add the new or renamed library to the front of the library list, then type **CHGPRODFT** on an AS/400 command line. The utility returns the current values and allows you to change them.

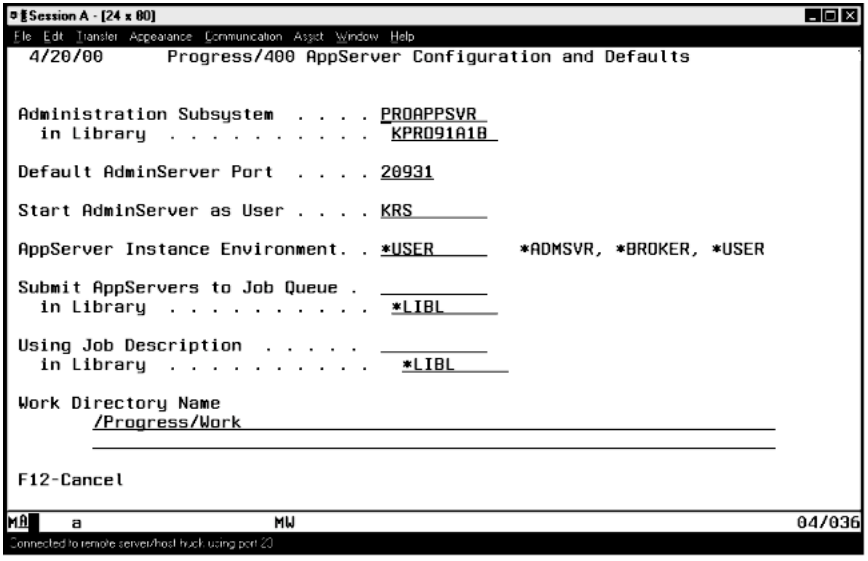
When you execute command, the following screens appear in succession. To move from one screen to the next press **ENTER**. To move to a previous screen press **F3**. When **ENTER** is pressed on the last screen, the changes are made:



You modify values in the Progress/400 Product Configuration screen to change values originally set during installation for the Progress/400 product:



You modify values in the Progress/400 Native-Client Configuration screen to change the temporary objects directory originally set during installation for the Native 4GL Client and the Progress/400 AppServer:



You modify values in the Progress/400 AppServer Configuration and Defaults screen to change the defaults originally set during installation for the Progress/400 AppServer.

10.4.3 Create Progress/400 Lock Table (CRTPROLKT)

Use the CRTPROLKT utility to create or modify a Progress/400 lock table. You can create only one lock table per Progress/400 server schema. Once created, the lock table exists until you explicitly delete it, whether or not there is an active Progress/400 session. The only modification you can make to an existing lock table is to change its number of entries.

For more information about Progress/400 lock tables, see the"Creating and Maintaining the Progress/400 Lock Table" section in Chapter 2, "Common Product Information."

Table 10–5 describes the CRTPROLKT parameters.

Table 10–5: CRTPROLKT Parameters

Parameter	Keyword	Value
Progress/400 Dictionary Library	PRODCT	Enter the name of a Progress/400 dictionary library. If a lock table is not associated with this library, the utility creates it, otherwise it modifies it as specified by the NUMLCK parameter.
Number of Lock Table Entries	NUMLCK	Enter an integer value from 100 to 99,999 that specifies the number of entries in the relevant Progress/400 lock table. A new lock table is created with this number of entries and an existing lock table is modified to contain this number of entries. The default is 500. You might have to adjust the value you set initially, depending on your number of users, to obtain maximum performance.

10.4.4 Create Schema Image (CRTSCHIMG)

Use the CRTSCHIMG utility to add a schema image to your existing dictionary library. If your dictionary library contains an existing schema image, this utility overwrites it. Once the new schema image is created, this utility synchronizes the schema image with the server schema.

Table 10–6 describes the CRTSCHIMG parameter.

Table 10–6: CRTSCHIMG Parameter

Parameter	Keyword	Value
Dictionary Library	DCTLIB	Enter the Progress/400 server schema library name. This utility automatically synchronizes the schema image with the server schema.

10.4.5 Convert Progress/400 Dictionary (CVTPRODCT)

Use the CVTPRODCT utility to convert any previous version of a Progress/400 Dictionary to the current version. You can run this utility only once against a Dictionary Library.

NOTE: You must run DUPPRODB to create an empty dictionary library before you run CVTPRODCT.

Table 10–7 describes the CVTPRODCT parameters.

Table 10–7: CVTPRODCT Parameters

Parameter	Keyword	Value
Source Dictionary Library	SRCDCTLIB	Enter the name of the library that contains the Progress/400 Dictionary that you want to convert.
Destination Dictionary Library	DSTDCTLIB	Enter the name of the library that contains the Progress/400 Dictionary that is the destination for the conversion. Progress Software recommends that you use a separate library for your server schema.
Overlay Source Dict. Schema	CVTSRCDCT	This additional parameter, which you access using the **F10** control key, specifies whether the Source Dictionary Library is converted in place. In a conversion in place, the Source Dictionary Library server schema is overlaid with new information. NOTE: **Use this option at your own risk.** You will be able to use the converted dictionary **only** with the current version of Progress/400. - Enter *YES to perform the conversion in place and overlay the Source Dictionary Library server schema with the current Progress/400 version server schema information. Note that if you enter *YES, you must specify identical names for the Source Dictionary Library and Destination Dictionary Library parameters. - Enter *NO (the default) to convert the Source Dictionary Library to a different Destination Dictionary Library; that is, to not perform the conversion in place.

10.4.6 Convert Server Schema (CVTSRVSCH)

Use the CVTSRVSCH utility to update an existing server schema when you install a new version of Progress/400.

Table 10–8 describes the CVTSRVSCH parameter.

Table 10–8: CVTSRVSCH Parameter

Parameter	Keyword	Value
Destination Dictionary Library	DCTLIB	Enter the name of the library that contains the server schema that you want to convert. Press + to enter additional Dictionary Libraries.

10.4.7 Duplicate Progress/400 Database (DUPPRODB)

Use the DUPPRODB utility to perform the following tasks:

- Create a Progress/400 server schema.
- Duplicate an existing Progress/400 server schema and DB2/400 database files, with or without data. See the "Creating Test and Production Environments" section in Chapter 8, "System Administration," for a detailed description of using the DUPPRODB utility to duplicate dictionaries.
- Create a schema image.

NOTE: All of the files in the dictionary that you want to copy should have *OBJEXIST, *OBJMGT, *OBJOPR, and *READ authority to the user performing DUPPRODB operations.

Table 10–9 describes the DUPPRODB parameters.

Table 10–9: DUPPRODB Parameters *(1 of 5)*

Parameter	Keyword	Value
New Progress/400 DB Name	NEWDB	Enter the name for the new library that the utility creates or *CURLIB to specify the current library.
From Progress/400 DB Name	FRMDB	Enter *PROEMPTY to duplicate an empty Progress/400 server schema in the new library. Enter the name of an existing Progress/400 server schema. Enter *PROSPORT or *PRODEMO to duplicate the server schema of the Progress demonstration database.

Table 10–9: DUPPRODB Parameters *(2 of 5)*

Parameter	Keyword	Value
Dictionary Copy Option	CPYOPT	This option appears only if you do not enter *PROEMPTY in the FRMDB option. If it appears, enter one of the following options: - Enter *FULLCPY to create the server schema in NEWDB and duplicate the objects defined in the server schema in the library specified at the Database Object Library parameter. The new objects contain the data stored in the existing objects. - Enter *EMPTYCPY to copy only the objects contained in a Progress/400 Data Dictionary and no data. It creates the server schema in the NEWDB from the FRMDB and the objects defined in the server schema. The new objects do not contain data. If you want the objects to be created in a library other than the library that contains the server schema, specify that library at the Database Object Library option. The utility creates the empty object in the OBJLIB and updates the references to that library in the server schema. - Enter *DCTONLY to copy only the server schema from the FRMDB to the NEWDB and no objects or data. If your NEWDB already has a server schema defined, enter *YES at the Overwrite Existing Dictionary option. If the utility encounters a failure, it backs out any duplications and restores the NEWDB library to its prior state.
Create Schema Image	CRTSCHIMG	Enter *YES to create a schema image. The schema image automatically synchronizes with the server schema. The default value for this parameter is *NO.

Table 10–9: DUPPRODB Parameters *(3 of 5)*

Parameter	Keyword	Value
Case Sensitive Altseq Table	ALTSEQ	Enter *NONE if the DB2/400 database files that you are accessing use the IBM037 code page. If the objects use any other code page, you must provide the names of the object files and libraries for the alternate collation and case tables. See Chapter 8, "System Administration," for a discussion of code pages and for descriptions of alternate sequence tables.
Case Insensitive Altseq Table	ALTSEQCI	Enter the name of the case-insensitive alternate collating sequence table. The case-insensitive alternate collating sequence table parameter ensures the case-insensitive behavior that is consistent with Progress. The OS/400 is case sensitive by default. Enter *NONE if you require case sensitivity.
Upper Case Table	UPCASE	Enter the name and library for the uppercase table that your DB2/400 database files use.
Lower Case Table	LOCASE	Enter the name and library for the lowercase table that your DB2/400 database files use.
Word Break Table	SEPTABLE	Enter the name and library of the word break table for your Progress/400 word indexes.

Table 10–9: DUPPRODB Parameters *(4 of 5)*

Parameter	Keyword	Value
DataServer Database Code Page	DBCODPAG	Enter the name of the code page that your DB2/400 database files use. By default, this parameter is set to *SYSVAL, the code page that your system uses. Your DB2/400 database files might use a different code page; enter its name here. The name must correspond to the name of the Case Sensitive Altseq Table on the AS/400 and to the name of a code page listed in the `CONVMAP.DAT` file on the client. If the code-page name is not listed, you can add an entry for it. The client handles all code-page conversions; it converts only those fields that an application requests. See the character-processing chapter in the *Progress Internationalization Guide* for instructions. See the *AS/400 National Language Support Planning Guide* for more information on code pages.
Dictionary Library Description	TEXT	Specify information to store in the P__DB description file.
Overwrite Existing Srvr Schema	OVRWRTDCT	Enter *YES to overwrite an existing server schema in a library with the same name as the new Progress/400 dictionary. This is the default. Enter *NO to not overwrite the existing server schema.

Table 10–9: DUPPRODB Parameters *(5 of 5)*

Parameter	Keyword	Value
Database Object Library	OBJLIB	Enter *NEWDB to use the same library to contain both the Dictionary and the database files. Enter a library name or enter *CURLIB to specify the current library. If the library exists, the utility uses it to contain the database files. If the library does not exist, the utility creates it and places the database files in it.
Create Progress Lock Table	CRTLKT	Enter *NO if you want OS/400 locking behavior. See Chapter 2, "Common Product Information," for a comparison of Progress and OS/400 locking. Enter *YES if you want locking behavior that is compatible with Progress locking. The default is 500 locks. If more locks are needed, use CRTPROLKT.

The utility creates the server schema in the Progress/400 Dictionary Library. It contains the OS/400 files listed in Table 10–10.

Table 10–10: Progress/400 Server Schema Files *(1 of 2)*

OS/400 File	Description
P__DB	Database definition file
P__FILE	File information file
P__FIELD	Field information file
P__INDEX	Index information file
P__IDXFL	Index key information file
P__SCHEMA*	Schema image files if CRTSCHIMG (*YES)
P__SEQ	Sequences information file

Table 10–10: Progress/400 Server Schema Files *(2 of 2)*

OS/400 File	Description
P__TRGFD	Field trigger information file
P__TRGFL	File trigger information file
P__USER	Currently not used
P__VCOL	Currently not used
P__VIEW	Currently not used
P__VREF	Currently not used

The utility also creates the OS/400 objects listed in Table 10–11.

Table 10–11: Progress/400 Server Schema Objects

Object Type	Name	Description
Journal	PRODBAJRN	Journal for P files
Journal receiver	PRODBA*xxxx*	Journal receiver for PRODBAJRN
User index	PRODBAIDX	DBA journal processing index
User space	PROLKT	Progress/400 lock table
User space	P__CTL	Dictionary control area

10.4.8 DUMP DATA (DMPPRODTA)

Use the DMPPRODTA utility to perform a native dump of data from one or more physical files in the Progress/400 dictionary library. The data is dumped into an output file in a specified directory in the root file system.

Note that this utility performs the dump **directly on the AS/400** as opposed to through client-server processes, as discussed in the "Dumping Data" section in Chapter 9, "Remote Client DB2/400 Utilities."

Table 10–12 describes the DMPPRODTA parameters.

Table 10–12: DMPPRODTA Parameters

Parameter	Keyword	Value
Dictionary Library	DCTLIB	Enter the name of the Progress/400 dictionary library containing the physical files to be dumped.
To Directory	TODIR	Enter the name of the directory in which to create the output dump file or *CURDIR (the default) to indicate the current directory.
Progress Table Name(s)	PROTBL	Enter the name of the Progress table to dump or *ALL (the default) to dump all tables in the Progress/400 dictionary library.

10.4.9 LOAD DATA (LODPRODTA)

Use the LODPRODTA utility to perform a native load of data into one or more physical files in the Progress/400 dictionary library. The data is loaded from an input file in a specified directory in the root file system.

Note that this utility performs the load **directly on the AS/400** as opposed to through client-server processes, as discussed in the "Loading Data" section in Chapter 9, "Remote Client DB2/400 Utilities."

Table 10–13 describes the LODPRODTA parameters.

Table 10–13: LODPRODTA Parameters

Parameter	Keyword	Value
Dictionary Library	DCTLIB	Enter the name of the Progress/400 dictionary library containing the physical files to be loaded.
From Directory	FROMDIR	Enter the name of the directory containing the input load file, or enter *CURDIR (the default) to indicate the current directory.
Progress Table Name(s)	PROTBL	Enter the name of the Progress table to load or *ALL (the default) to load all tables in the Progress/400 dictionary library.

10.4.10 Remove Server Schema Entry (RMVSCHE)

Use the RMVSCHE utility to remove schema entries from a server schema. This utility allows you to remove entries from the Progress/400 environment's physical, virtual, and logical files without removing them from the selected library. After you run this utility, you must synchronize the clients so that the schema holders reflect the changes.

The "Usage Guidelines for the RMVSCHE Utility" section provides some special information on using this utility. Read it before you run this utility. For more information on how the RMVSCHE utility executes, see the "Operational Notes for the RMVSCHE Utility" section.

Table 10–14 describes the RMVSCHE parameters.

Table 10–14: RMVSCHE Parameters *(1 of 2)*

Parameter	Keyword	Value
Dictionary Library	DCTLIB	Enter the name of the library containing the server schema to be synchronized.
Server Schema Entry	ENTRY	Enter a list of from 1 to 25 entries to remove from the server schema. You must include the following information for each entry: *entry-name*: The file name of the entry to remove. If you specify an entry type of *TABLE, you can specify either *entry-name* or *ALL. *library*: The name of the library where the file resides. If you specify an entry type of *TABLE, you can specify either *library* or *ALL. *entry-type:* The entry type; that is, the type of file specified for this entry. The possible values are: *TABLE: Any file entry in the P__FILE file *INDEX: Any file entry in the P__INDEX file *RELATIONS: All entries in the P__INDEX file for the named file except physical file key

Table 10–14: RMVSCHE Parameters *(2 of 2)*

Parameter	Keyword	Value
Automatically Reassign Primary Index	ASGNPRIM	Specify whether to reassign the primary index automatically. Enter one of the following: - Enter *YES (the default) to automatically reassign the primary index. - Enter *NO to not automatically reassign the primary index.
Synchronize Schema Image	SYNCSCHE	Specify whether to synchronize the schema image. Enter one of the following: - Enter *YES to synchronize the schema image. This option allows you to remove items from the server schema and update the schema image in one step. - Enter *NO (the default) to not synchronize the schema image.

Usage Guidelines for the RMVSCHE Utility

You must follow these rules when you use the RMVSCHE utility:

- You can specify *ALL values only when the entry type is *TABLE.
- The Automatically Reassign Primary Index parameter works in conjunction with the *INDEX option.
- You cannot remove the physical file key either by explicitly naming the file or by selecting the *RELATIONS option.
- Setting the Automatically Reassign Primary Index parameter to *NO and entering the primary index name in the file name parameter does not remove the primary index.
- The *RELATIONS option does not use the Automatically Reassign Primary Index parameter.

Operational Notes for the RMVSCHE Utility

This section provides additional information on how the RMVSCHE utility operates:

- If you specify *ALL for both the filename and the library name and use *TABLE, the server schema is cleared.
- If you specify *YES for the Automatically Reassign Primary Index parameter, the first index found in the P__INDEX file becomes the new primary index.
- The utility lists the files removed from the server schema in the job log.
- If an error occurs, a rollback is performed. This means that either all changes or no changes are processed.
- The validity checker provides the normal checking to verify access and, in addition, validates the following:
  - The Dictionary Library is correct.
  - The server schema version is correct.
  - The named files are in the server schema.
  - If the *ALL option is used for the file name and/or the library name, the entry-type is *TABLE.

  NOTE: If the *ALL option is used with any entry other than *TABLE, an error message is returned.

10.4.11 Synchronize Schema Image (SYNSCHIMG)

Use the SYNSCHIMG utility to synchronize the schema image with the server schema. SYNSCHIMG copies the P__FILE definitions from the server schema into the schema image.

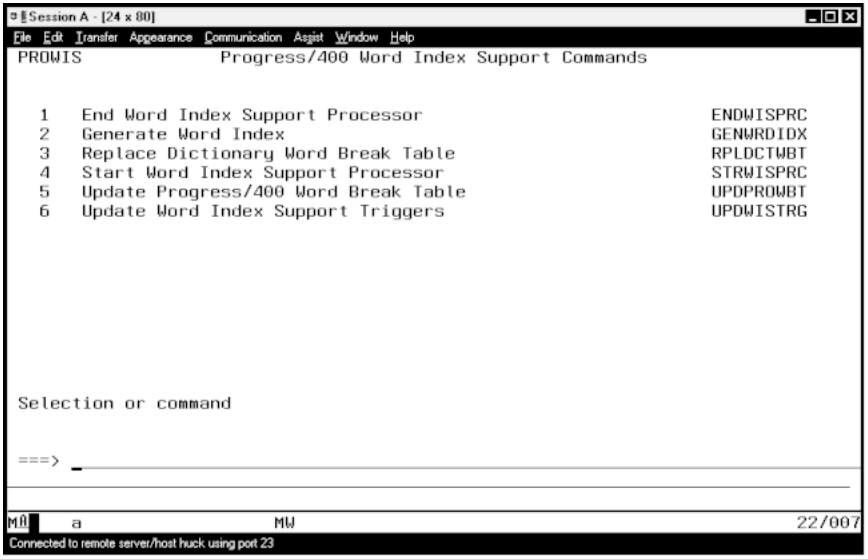
Table 10–15 describes the SYNSCHIMG parameter.

Table 10–15: SYNSCHIMG Parameter

Parameter	Keyword	Value
Dictionary Library	DCTLIB	Enter the name of the dictionary library that contains the schema image.

10.5 Progress/400 Word Index Support Commands

This section explains when to use the Progress/400 word index utilities and defines the fields within each utility. When you select Progress/400 Word Index Support Commands from the Progress/400 Main Menu screen, the following menu appears:



Type the number of the utility to run after the arrow, then press **RETURN**.

10.5.1 End Word Index Support Processor (ENDWISPRC)

Use the ENDWISPRC utility to stop the Word Index Support Processor jobs. For a description of word indexing, including the Word Index Support Processor, see the "Progress/400 Word Index Support" section in Chapter 2, "Common Product Information."

The ENDWISPRC utility has no parameters.

10.5.2 Generate Word Index (GENWRDIDX)

Use the GENWRDIDX utility to rebuild one or all word indexes associated with a physical file. A word index that is to be generated **must not** be locked by any other AS/400 job. This utility requires exclusive use of any index that it is to change.

For a description of word indexing, see the "Progress/400 Word Index Support" section in Chapter 2, "Common Product Information."

Table 10–16 describes the GENWRDIDX parameters.

Table 10–16: GENWRDIDX Parameters

Parameter	Keyword	Value
Dictionary Library	DCTLIB	Enter the name of the Progress/400 Dictionary Library.
Physical File	FILE	Enter the name of the physical file and library on which Progress/400 word indexes are built.
Word Index	WRDIDX	Enter the name and library of the word index to generate or *ALL to specify that all Progress/400 word indexes for the specified physical file are rebuilt.

10.5.3 Replace Dictionary Word Break Table (RPLDCTWBT)

Use the RPLDCTWBT utility to change the word break table name for a Progress/400 Dictionary Library. If word indexes exist over physical files in the Dictionary Library, they are rebuilt using the new word break table.

For a description of word indexing, including word break tables, see the "Progress/400 Word Index Support" section in Chapter 2, "Common Product Information."

Table 10–17 describes the RPLDCTWBT parameters.

Table 10–17: RPLDCTWBT Parameters

Parameter	Keyword	Value
Dictionary Library	DCTLIB	Enter the name of the Progress/400 Dictionary Library.
Word Break Table Name	WBT	Enter the name and library of the Progress/400 word break table.

10.5.4 Start Word Index Support Processor (STRWISPRC)

Use the STRWISPRC utility to start the Word Index Support Processor. This Processor consists of two perpetual jobs named PROWISRCV and PROWISDTAQ. When you run STRWISPRC, it submits the PROWISRCV job to the job queue. Once PROWISRCV starts, it starts the PROWISDTAQ job.When these jobs have started, each executes a CHGJOB command on itself

to change its Run Priority (RUNPTY) to 20 and its Time Slice (TIMESLICE) to 2000. This forces both jobs to run at interactive priority, which ensures that they run very quickly.

If the PROWISDTAQ job ends abnormally because of an error or because an ENDJOB command is run against it, the PROWISRCV job starts it again. This ensures that the PROWISRCV job can perform its task. However, if the PROWISRCV job ends, you must restart this job with the STRWISPRC utility.

There is exactly one PROWISRCV job per Word Index Work Library, and each PROWISRCV job starts exactly one PROWISDTAQ job. If the Word Index Support Processor jobs are already running when you execute the STRWISPRC utility, the utility returns a message noting that only one set of Processor jobs can be active at one time.

When the STRWISPRC utility executes, it deletes old journal receivers and reorganizes the PROTRGTXN file. You should shut down and restart the Word Index Support Processor regularly to minimize the number of old receivers and the size of the PROTRGTXN file.

For a description of word indexing, including the Word Index Support Processor and the Word Index Work Library, see the "Progress/400 Word Index Support" section in Chapter 2, "Common Product Information."

Table 10–18 describes the STRWISPRC parameters.

Table 10–18: STRWISPRC Parameters

Parameter	Keyword	Value
Submit to Job Queue	JOBQ	Enter the name of the job queue where you submit the Word Index Support Processor's PROWISRCV job.
In Library	N/A	Enter the name of the library where the job queue exists.

10.5.5 Update Progress Word Break Table (UPDPROWBT)

Use the UPDPROWBT utility to create or update a word break table. When you run this utility, a screen appears. You create or update the word break table by modifying the information in this screen. See the "UPDPROWBT Screen" section for details.

For a description of word indexing, including word break tables, see the "Progress/400 Word Index Support" section in Chapter 2, "Common Product Information."

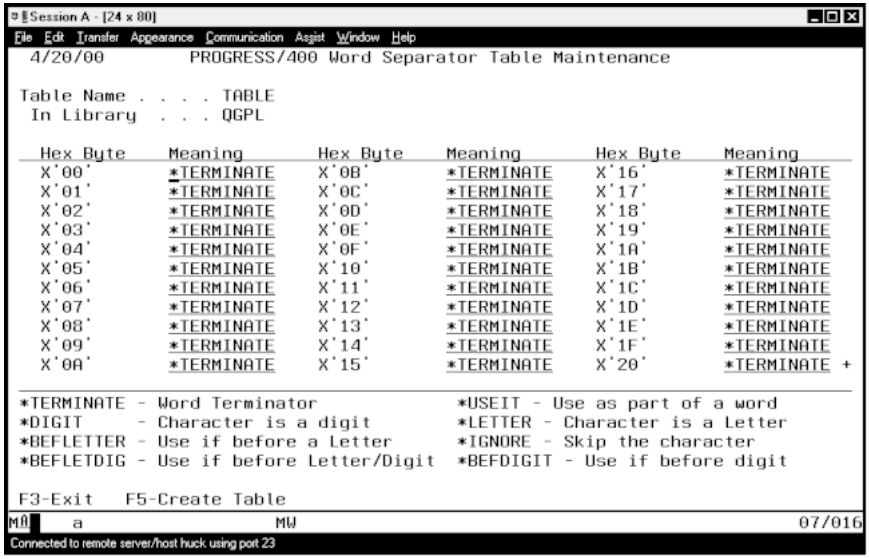
Table 10–19 describes the UPDPROWBT parameters.

Table 10–19: UPDPROWBT Parameters

Parameter	Keyword	Value
Word Break Table Name	WBT	Enter the name of the word break table to create or maintain. If the table does not exist, the utility creates it. If it does exist, the utility maintains it.
Library Name	N/A	Enter the name of the library where the word break table exists or where it is to be created or *LIBL to specify the library list. If you enter *LIBL and the table is not found in the job's library list, it is created in the job's *CURLIB (current library).

UPDPROWBT Screen

The screen that appears when you run the UPDPROWBT utility looks like this:



You modify values in this screen to define the actual contents of the word index table. The columns contain the following:

- The Hex Byte column contains a character for which a meaning is being defined.
- The associated Meaning column contains a value that indicates the meaning.

For example, Progress/400 word index support treats the hexadecimal character X'00'as a terminator.

10.5.6 Update Word Index Support Triggers (UPDWISTRG)

Use the UPDWISTRG utility to change the library of the trigger programs for physical files on which word indexes are built.

Suppose, for example, that you create word indexes over a file using Progress/400 in the library PROGRESS, then change the name of the library from PROGRESS to PROV80C40. You must now run this utility with the new library in your library list. After the utility finishes executing, the triggers point to the programs in the correct library. If you make a change of this type but do not run this utility, you cannot open those physical files.

For a description of word indexing, see the "Progress/400 Word Index Support" section in Chapter 2, "Common Product Information."

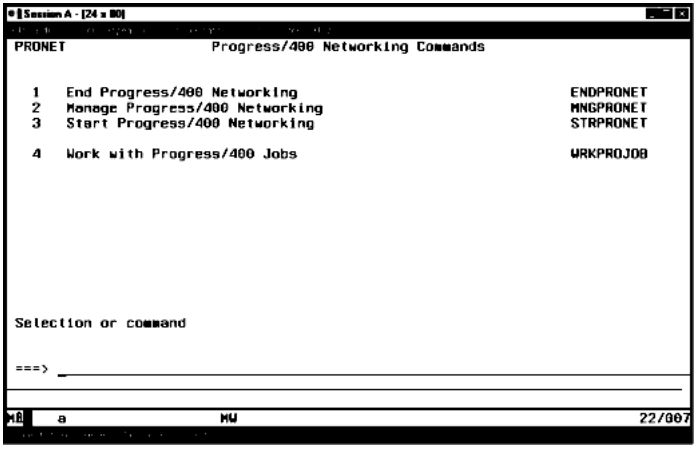
Table 10–20 describes the UPDWISTRG parameter.

Table 10–20: UPDWISTRG Parameters

Parameter	Keyword	Value
Dictionary Library(s)	DCTLIB	Enter the name of one or more Progress/400 Dictionary Libraries.

10.6 Progress/400 TCP/IP Network Commands

This section explains when to use the Progress/400 networking utilities and defines the fields within each utility. When you select Progress/400 TCP/IP Network Commands from the Progress/400 Main Menu screen, the following menu appears:



Type the number of the utility to run after the arrow, then press **RETURN**.

10.6.1 End Progress Network (ENDPRONET)

Use the ENDPRONET utility to stop the broker for TCP/IP.

Table 10–21 describes the ENDPRONET parameters.

Table 10–21: ENDPRONET Parameters

Parameter	Keyword	Value
Protocol	PROTOCOL	Enter the name of the networking protocol. The only possible value is *TCP, which specifies the TCP/IP protocol.
TCP Information	TCPINF	Enter the name of the TCP/IP Broker Service or port number to end. Specify the same value that you specified for the STRPRONET utility.

10.6.2 Manage Progress/400 Networking (MNGPRONET)

Use the MNGPRONET utility to manage, monitor the status of, or stop a TCP/IP broker. Running the MNGPRONET utility provides a list of all available TCP/IP broker processes and a list of tasks to perform. Determine the appropriate TCP/IP broker, then select the task that you want to perform on the broker. See the "Managing and Stopping TCP/IP Brokers" section in Chapter 4, "Remote Access to Progress/400 DataServer," for usage information.

Table 10–22 describes the MNGPRONET parameters.

Table 10–22: MNGPRONET Parameters

Parameter	Keyword	Value
Protocol	PROTOCOL	Enter the name of the networking protocol. The only possible value is *TCP, which specifies the TCP/IP protocol.

10.6.3 Start Progress/400 Networking (STRPRONET)

Use the STRPRONET *TCP utility to start a TCP/IP broker.

Table 10–23 describes the STRPRONET parameters for TCP/IP.

Table 10–23: STRPRONET *TCP Parameters *(1 of 2)*

Parameter	Keyword	Value
Protocol	PROTOCOL	Enter *TCP.
Broker/Manager Job Queue	SBMQUE	Enter the name of the job queue that you want to assign to the TCP/IP broker. If other jobs are required to execute concurrently through the same job queue, the job queue should be multi-threaded. See the *AS/400 Work Management Guide* for information on working with job queues.
TCP Information	TCPINF	Specify line and connection information.
Service Name	N/A	Enter the name of the service or port number on the AS/400 that the TCP/IP broker uses. Use the CFGTCP CL command (option 21) to look up the name in TCP Services Name.

Table 10–23: STRPRONET *TCP Parameters *(2 of 2)*

Parameter	Keyword	Value
Server Job Queue	N/A	Enter the name of the job queue that you want to assign to the database server. If other jobs are required to execute concurrently through the same job queue, the job queue should be multi-threaded. See the *AS/400 Work Management Guide* for information on working with job queues.
Library	N/A	Enter the name of the library that contains the job queue, the value *LIBL to specify your library list, or the value *CURLIB to specify the current library.
Server Timeout	N/A	Specify the number of seconds that the server waits before retrying to connect after a failed response. Enter a value from 30 and 180. The default is 60.
Start Server w/ Message Logging	N/A	Specify whether to start the server with message logging. Enter one of the following: - Enter *NO (the default) to not log messages. - Enter *YES to log messages.
First Server Port to Assign	N/A	Specify the number of the first server port to assign. The default is 1025.
Last Server Port to Assign	N/A	Specify the number of the last server port to assign. You must specify a value that is greater than the value of the First Server Port to Assign parameter. The default is 2000.

10.6.4 Work with Progress Jobs (WRKPROJOB)

Use the WRKPROJOB utility to display information about active Progress/400 jobs. You can display information about TCP/IP brokers, SNA or TCP/IP servers, or native clients. The initial display of information lists all active jobs of the specified type and provides certain types of job information about each

This utility also provides limited job-control functionality. You can use it to end server and client processes. You cannot use it to end TCP/IP broker processes; use either MNGPRONET or ENDPRONET instead. See the"Manage Progress/400 Networking (MNGPRONET)" and "End Progress Network (ENDPRONET)" sections for details.

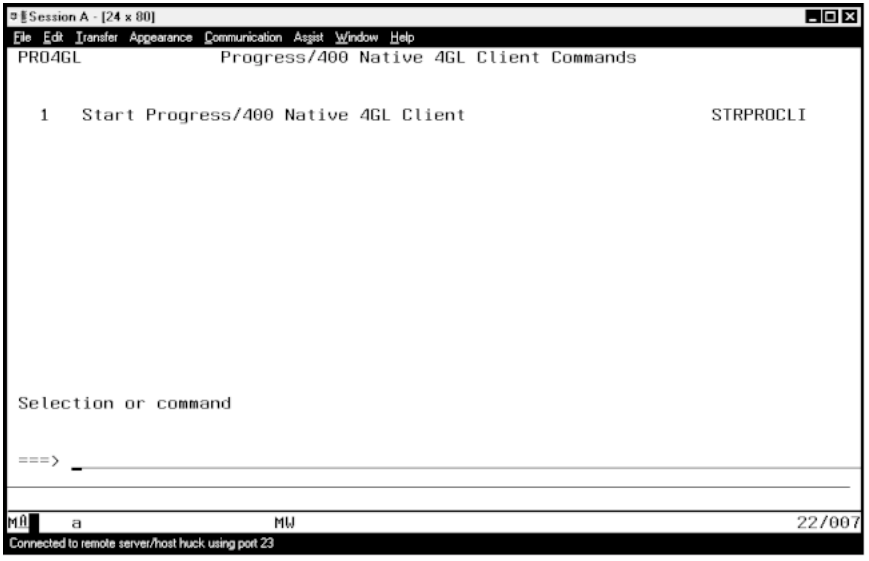
Table 10–24 describes the WRKPROJOB parameter.

Table 10–24: WRKPROJOB Parameter

Parameter	Keyword	Value
Progress/400 Job Type	TYPE	Specify the types of processes to list. Enter one of the following: - Enter *BROKER to list all TCP/IP broker processes. - Enter *SERVER to list all SNA and TCP/IP server processes. - Enter *CLIENT to list all native client processes.

10.7 Progress/400 Native 4GL Client Commands

This section describes the command for starting the Native 4GL Client. When you select Progress/400 Native 4GL Client Commands from the Progress/400 Main Menu screen, the following menu appears:



Type the number of the utility to run after the arrow, then press **RETURN**.

10.7.1 Start Native 4GL Client (STRPROCLI)

Use the STRPROCLI utility to execute Progress 4GL code. To execute this utility, you must provide the Progress 4GL code you wish to execute. All other parameters can be optional, depending on your specific environment needs.

Table 10–25 describes the STRPROCLI parameters.

Table 10–25: STRPROCLI Parameters *(1 of 2)*

Parameter	Keyword	Value
Startup Procedure (-p)	STRPROC	Specify Progress 4GL code that the Native 4GL Client will execute. This parameter is required.
Database Name (-db)	SCHDB	Enter the name of the library that contains your server schema.
PROPATH Value	PROPATH	Enter a list of directory paths separated by commas. When you run a procedure, Progress searches for the procedure in the directory paths defined in the PROPATH in the order listed.
Working Directory	WRKDIR	Enter the directory to which Progress defaults for current working directory files.
Print File (-o)	PRTF	Enter the printer to use when processing the OUTPUT TO PRINTER statement in procedures.
Parameter File Name (-pf)	PRMFMBR	Enter the name of a parameter file that contains Progress startup parameters.
Parameter String (-param)	PRMSTR	Enter a character string that supplies information to the 4GL application.
Progress Parameters	PROPARMS	Enter any additional Progress parameters that you want to specify. You must use standard Progress syntax: param -value -param -value

Table 10–25: STRPROCLI Parameters *(2 of 2)*

Parameter	Keyword	Value
Fast Heap Size in KB	FASTHEAP	Specify how OS/400 manages memory during this session of the Native 4GL Client. Enter one of the following: - Enter *DEFAULT to allocate the default fast heap size chosen by IBM. This was the behavior in previous Progress/400 releases. - Enter *NONE to not allocate a fast heap. The C program's heap is used instead. - Enter an integer value from 64 to 15,000 to allocate a specific amount of fast heap.
Log Client Messages	LOGMSG	Enter *YES to log client messages. Enter *NO (the default) to not log client messages.

You can provide additional parameters to the Native 4GL Client with the STRPROCLI utility. There are two ways to do this:

- With the PROPARMS option
- With a parameter file (-pf) using the PRMFMBR option

Table 10–26 lists an effective and useful subset of parameters that the Native 4GL Client supports.

Table 10–26: Startup Parameters for STRPROCLI *(1 of 2)*

Parameter	Description
Case Code Page (-cpcase)	Name of a case table within the CONVMAP.CP file.
Database Code Page (-cpdb)	Name of database code-page table.
Print Code Page (-cpprint)	Name of code page for printer output.
R-code in Code Page (-cprcodein)	Name of code page for reading r-code segments.

Table 10–26: Startup Parameters for STRPROCLI *(2 of 2)*

Parameter	Description
Stream Code Page (-cpstream)	Name of code page for stream I/O.
Physical Database Name (-db)	Name of the database to connect to when a Progress session starts.
DataServer (-Dsrv)	Keyword signifying that DataServer parameters follow.
Nested Blocks (-nb)	The maximum number of nested blocks.
Startup Procedure (-p)	The name of the procedure to run when starting Progress.
Parameter (-param)	A character string that provides information to the 4GL application.
Parameter File (-pf)	The name of a parameter file that contains startup parameters to run Progress.
Alternate Random Number Generator (-rand)	The type of random number generator.
Stack Size (-s)	The size of the stack in 1K units.
Merge Number (-TM)	The number of blocks or streams to be simultaneously merged during the sort process.

For detailed descriptions of these parameters, see the *Progress Startup Command and Parameter Reference*.

11

Progress 4GL Interfaces to OS/400 Languages and Objects

The Progress/400 client can access OS/400 commands and programs. It can submit OS/400 jobs, execute OS/400 commands, and execute special Progress/400 commands. Additionally, the Progress/400 product allows you to call OS/400 programs from within a Progress/400 procedure. The Progress/400 product also supports Progress 4GL access to data areas, user spaces, and data queues. This chapter covers these features within the following topics:

- External Programming Interfaces (EPI)
- Executing OS/400 commands
- Stored procedure support
- OS/400 object access

11.1 External Programming Interface (EPI)

When you write a Progress 4GL program using the Progress/400 Native 4GL Client, the External Programming Interface (EPI) gives you more flexibility when dealing with OS/400. The Progress OS400 statement has an EPI extension that provides a simple interface to OS/400 programs without affecting the core Progress 4GL.

11.1.1 General Syntax

This section describes the syntax of EPI:

```
OS400 EPI STATUS(sts) MESSAGES(msg) command command-parameters
```

Though OS400 is a Progress 4GL statement, all additional characters are treated as the EPI command. Progress processes the EPI command at run time. The STATUS(*sts*) and MESSAGES(*msg*) parameters allow the Progress 4GL to examine error messages returned from the EPI command.

The Progress 4GL uses the STATUS(*sts*) parameter as a status indicator (where *sts* is a Progress shared variable of type INTEGER). The Progress 4GL uses the MESSAGES(*msg*) parameter to store the error messages encountered during parsing or execution of the EPI command (where *msg* is a Progress shared variable of type CHARACTER with at least one extent). If the EPI command is successfully parsed and executed, the value in the *sts* variable is zero (0). If errors are encountered during parsing or execution of the EPI command, the value in the *sts* variable is greater than zero (0) and indicates how many messages have been placed in the *msg* array.

Use the OS-ERROR function to determine the status of the EPI command. Progress has extended this function to support new Progress/400 EPI error numbers. To determine the results of an EPI command, examine the value returned by the OS-ERROR function.

Table 11–1 describes the error codes that can be returned.

Table 11–1: OS-ERROR Values for EPI Command *(1 of 2)*

OS-ERROR Value	Error Description
4001	The STATUS variable must be defined as an integer.
4002	The MESSAGES variable must be defined as a character.
4003	The MESSAGES variable must be defined as an array with at least one EXTENT.

Table 11–1: OS-ERROR Values for EPI Command *(2 of 2)*

OS-ERROR Value	Error Description
4004	The STATUS parameter must be specified.
4005	The MESSAGES parameter must be specified.
4006	Invalid EPI command.
4007	Invalid syntax.
4999	An error occurred. The message text is in the MESSAGES variable.

11.1.2 EPI CALL Command

The EPI CALL command allows the Progress 4GL to call programs written in languages supported by OS/400. A called program can receive data in parameters and can return data to a Progress 4GL program in the same parameters.

The following restrictions apply to any OS/400 program called from a Progress 4GL program:

- The called program must expect parameters passed by reference. Specifically, the called program must accept an address that points to memory that contains the value of the parameter. For example, CL, RPG, and C programs all expect parameters passed by reference. MI programs can handle parameters passed by value and by reference. Any program that expects parameters passed by value cannot be called by the Progress 4GL.
- A maximum of 32 parameters can be passed to any called program.
- Parameters of the following OS/400 data types are supported:
  - CHARACTER
  - DECIMAL or PACKED
  - ZONED
  - LOGICAL as CHARACTER
  - INTEGER (four byte integer)

- The following Progress data types are supported:
  - DECIMAL
  - CHARACTER
  - INTEGER
  - LOGICAL

For details on data type support, see the "PARM Parameter" section.

Syntax

The syntax for the EPI CALL command is designed for use by both the Progress programmer and the OS/400 programmer. The EPI CALL syntax mixes both Progress and CL constructs, making it familiar to both types of programmers:

```
OS400 EPI STATUS(sts) MESSAGES(msg)
CALL PGM(library/program)
PARM(prog-var[extent] AS TYPE(len, decimals) USE use)
```

PGM Parameter

The CALL statement uses the PGM parameter to specify the name of the program the 4GL program is calling. The value and use of this parameter match those of the PGM parameter on the CL CALL command. You can explicitly specify the library or use the *LIBL. The PGM parameter is required. If you do not specify a library, the Progress 4GL assumes *LIBL. If the program or library is not found, the Progress 4GL places an error message into the MESSAGES variable.

PARM Parameter

The PARM parameter defines what parameters, if any, Progress passes to the called program and how Progress passes them. Any Progress variable used as a parameter for an HLL program must be defined as a SHARED VARIABLE. Defining variables in this fashion allows the value of the variable to be retrieved and changed efficiently. Progress shared variables are passed to an HLL program as a particular data type. The value *prog-var* contains the name of a Progress variable. The variable must be defined as a shared variable. A single element of a 4GL array (EXTENT field) can also be used as *prog-var*.

The Progress variable must be one of the following Progress data types:

- DECIMAL
- CHARACTER
- INTEGER
- LOGICAL

Progress requires the AS keyword. This states that the Progress variable will be passed to the called program as TYPE, where type indicates an OS/400 data type. The following OS/400 data types are supported:

- CHARACTER or CHAR
- DECIMAL or PACKED
- ZONED
- LOGICAL as CHARACTER
- INTEGER (4-byte integer)

Each parameter must have an OS/400 data type and a length. When you define the OS/400 data type, also define its length. To do this, use TYPE(len, decimals), where len indicates the length of the program parameter; and decimals, when required, specifies the number of decimal digits. Packed and Zoned parameters must have the number of decimals defined.

Table 11–2 indicates the rules to follow when defining the mapping between Progress and OS/400 data types.

Table 11–2: Mapping Progress to OS/400 Data Types *(1 of 2)*

Progress Data Type	OS/400 Data Type	Rules for Mapping
Character	Character	Data is passed as it appears in the Progress variable up to the length of the OS/400 data type.

Table 11–2: Mapping Progress to OS/400 Data Types *(2 of 2)*

Progress Data Type	OS/400 Data Type	Rules for Mapping
Integer	Integer	The value of the integer is passed to the HLL program as a four-byte integer field. Overflow could occur if a large number is passed back from the HLL program.
	Decimal Packed Zoned	The integer is converted to either packed or zoned of the specified length and decimal digits. However, the decimal portion of the packed or zoned field is set to 0 when the HLL program is called. The value of the decimal portions of the number from the HLL program is ignored when converting the value back into a Progress integer.
Decimal	Decimal	Data is passed as it appears in the Progress variable up to the length of the OS/400 data type.
Logical	Character	Data is passed as 1 or 0. A 1 is passed if the value is TRUE. A 0 is passed if the value is FALSE.

The USE parameter indicates how the Progress 4GL program treats the value of the variable. Table 11–3 describes the USE values.

Table 11–3: USE Parameter

USE Value	Description
INPUT	The Progress variable is treated as input to the called program and its value does not change in the 4GL program.
INPUT-OUTPUT	The Progress variable is treated as input to the called program. When the called program returns, the Progress variable reflects what is passed back from the program.
OUTPUT	The called program is called with a default value. When the called program returns, the Progress variable reflects what is passed back from the program. If the OS/400 data type is CHARACTER, blanks are passed. If the OS/400 data type is numeric, a value of zero (0) is passed.

The following code shows how to construct the EPI command for a call to a CL program:

Progress 4GL Program TestCall.P

```
DEFINE NEW SHARED VARIABLE Parameter1 AS CHARACTER FORMAT "X(50)".
DEFINE NEW SHARED VARIABLE Parameter2 AS DECIMAL FORMAT "999.9999".
DEFINE NEW SHARED VARIABLE RTN-STAT AS INTEGER.
DEFINE NEW SHARED VARIABLE Messages AS CHARACTER EXTENT 50.

ASSIGN Parameter1 = "This is a test"
ASSIGN Parameter2 = 3.14156.

DISPLAY Parameter1 Parameter2.

OS400 EPI STATUS(RTN-STAT) MESSAGES(Messages)
  CALL PGM(*LIBL/TESTCLP)
    PARM(Parameter1 AS CHARACTER(50) USE INPUT-OUTPUT)
    PARM(Parameter2 AS PACKED(15, 5) USE OUTPUT).

DISPLAY Parameter1 Parameter2.
```

CL Program TESTCLP

```
PGM PARM(&TEXT &NUMBER)

DCL VAR(&TEXT) TYPE(*CHAR) LEN(50)
DCL VAR(&NUMBER) TYPE(*DEC) LEN(15 5)

/* Alter the values of the parameters and return to the caller */

CHGVAR VAR(&TEXT) VALUE('to show that it works')
CHGVAR VAR(&NUMBER) VALUE(123.5623)

RETURN
ENDPGM
```

Example Progress 4GL programs and OS/400 programs are provided with your release. Sample HLL programs are available in the Examples Source File in the Progress-supplied product library.

11.2 Executing OS/400 Commands

The Progress/400 DataServer performs these functions from within a Progress client session:

- Submits OS/400 jobs
- Executes OS/400 commands
- Executes special Progress/400 commands

When you create a client schema holder, the DataServer loads a data definition file called QCMD, which provides this functionality. QCMD contains a parameter form for the OS/400 Submit Job (SBMJOB) command. When you complete the parameter form and press **F2**, the DataServer either performs the SBMJOB function, processes the OS/400 command, or processes the special Progress/400 command. The entry in the CMD parameter, which is the first entry on the parameter form, determines how the DataServer handles the request.

The Progress client session is not notified of the job's command status or of errors or warnings that the job generates. Use the OS/400 Display Job (DSPJOB) command to view job information.

Before you can submit a job, you must connect to the client schema holder and the DB2/400 database files.

11.2.1 QCMD Functions

Two techniques are available for submitting OS/400 commands from a Progress client:

- If the value for the CMD parameter is an OS/400 command, the Progress/400 DataServer issues an SBMJOB command using the entries that you provide on the parameter form. (It uses the SBMJOB command default values if you do not specify an entry is for a parameter on the form.) See Table 11–4 for a list of SBMJOB parameters and default values.
- If the first character for the CMD parameter is an exclamation point (!) or an asterisk (*), and the second character is a space, the Progress/400 DataServer executes the command immediately. (The command begins on the third character of the entry.)

  NOTE: You must use an asterisk instead of an exclamation point for all DataServers whose code pages do not translate to the exclamation point in the same way as code page IBM037.

QCMD Field Size Limitation

The field cmd in the QCMD file has a maximum length of 256 characters. However, a format of x(60) has been defined for the field cmd in the as4empty.df file. This allows entering only 60 characters when an UPDATE cmd is performed. The solution to this limit is to use the following p-code to allow entry of up to 256 characters into the cmd field:

```
DO TRANSACTION:
  CREATE QCMD.
  UPDATE cmd VIEW-AS EDITOR.
    INNER-CHARS 50 INNER-LINES 5 MAX-CHARS 256
    LABEL "Command".
  VALIDATE  QCMD.
END.
```

The following p-code will fill the field cmd with all characters in the assign statement:

```
DO TRANSACTION:
  CREATE QCMD.
  ASSIGN cmd = "! CRTMOD MODULE(QTEMP/ASNSSTOP) SRCFILE(MYLIB/SR) " +
    "SRCMBR(ASNSSTOP) OUTPUT(*PRINT) OPTION(*SYSINCPATH) " +
    "DBGVIEW(*ALL)"
  VALIDATE  QCMD.
END.
```

The Progress/400 DataServer supports a special command for opening and closing files.

- If the entry for the CMD parameter is !CLOSE *ALL, the DataServer closes all of the open database files.
- If the entry is !CLOSE *xxx* (where *xxx* is the name of an OS/400 file), Progress/400 closes the named file in the connected database (if the file was open).

NOTE: For the CLOSE commands, there is no space between the exclamation point (!) and the CLOSE keyword.

If the entry is not one of these cases, Progress/400 writes a message to the job log and does not perform any processing.

When you use QCMD for the SBMJOB command, Progress/400 submits the command to the job queue specified for the JOBQ parameter, where it runs under the user profile specified for the USER parameter. When you use QCMD to process a special Progress/400 command, the DataServer immediately processes it and ignores the remaining SBMJOB parameters.

11.2.2 Running RPG Programs with QCMD

You can run RPG programs using QCMD. However, QCMD cannot pass parameters to an RPG program, nor can it return status messages. To run RPG using QCMD, you must define a file that contains a status record to Progress.

11.2.3 Executing the SBMJOB Command

You can use either the Progress INSERT or CREATE statements to perform a QCMD function. The INSERT statement accesses the parameter form for the SBMJOB command. The parameter form is useful when you want to perform the SBMJOB function and are not familiar with all of the parameters. The CREATE statement is useful when you are familiar with the SBMJOB parameters or if you want to run OS/400 commands other than SBMJOB.

Accessing the QCMD Parameter Form

To access the QCMD parameter form using the INSERT statement, connect to the client schema holder and the DB2/400 database files, then follow these steps:

1 ♦ Access the Progress Procedure Editor.

2 ♦ Enter the following statement:

```
INSERT QCMD WITH 2 COL.
```

3 ♦ Press **GO**. Progress displays the parameter form for the OS/400 Submit Job (SBMJOB) command.

4 ♦ Complete the parameters that are necessary for the job you are submitting, then press **GO**. Progress/400 submits the job to the queue that you specify for the JOBQ parameter.

Running SBMJOB Without the QCMD Parameter Form

To issue an OS/400 command using the CREATE statement, connect to the client schema holder and the DB2/400 database, then follow these steps:

1 ♦ Access the Progress Procedure Editor.

2 ♦ Enter the following statement:

```
DO TRANSACTION.
  CREATE QCMD.
  ASSIGN cmd = "QCMD function"
    job = "MYJOBNAME"
    jobq = "QBATCH".
END.
```

In this statement, the QCMD function is one of the two special Progress commands, or a valid OS/400 CL command optionally prefixed by an exclamation point or an asterisk and followed by a space. MYJOBNAME and QBATCH are the job name and job queue, respectively, that were selected for submitting the command.

3 ♦ Press **GO**. The DataServer performs the QCMD function.

NOTE: If you are connected to multiple databases and the command you want to execute is specific to a database (for example, you want to close all PRODEMO files), you must qualify the QCMD file with the database name when you perform the INSERT or CREATE statement. For example, CREATE PRODEMO.QCMD.

QCMD can be useful when you use the Override Database File (OVRDBF) command for a file in your Progress/400 database with multiple members. All members must share the same physical file format. If you override the database member to a file format that is different from the default file member, the DataServer generates unpredictable results at run time.

Feedback Information

When the DataServer executes a native CL command using the QCMD function, the job number of the OS/400 job that executes the QCMD command can be returned by the Progress record ID (RECID). The following example demonstrates how to display the returned job number:

```
DEFINE VARIABLE crecid AS RECID.
CREATE QCMD.
ASSIGN cmd = "* qcmd function"
       crecid = RECID(QCMD).
DISPLAY crecid.

CREATE QCMD.
ASSIGN cmd = "qcmd function"
       job  = "MYJOBNAME"
       jobq = "QBATCH".

       crecid = RECID(QCMD).
DISPLAY crecid.
```

When you execute the QCMD command with the leading exclamation point or asterisk followed by the space on the cmd field, the DataServer job executes the command directly. If the leading special character is a plus (+), the value that is returned to the variable crecid is the job number of the DataServer. If you do not specify a leading special character on the cmd field, the DataServer submits a job to the operating system to perform the command. In this case, the value returned to the variable crecid corresponds to the job number of the submitted job. A return value of zero in crecid indicates that the DataServer failed to execute or submit the command request.

Error Handling

If there is an error on the command that you submit and you are using the Native 4GL Client, the error message is in the user's job log. If you are on a remote client, the error message is in the server's job log.

11.2.4 Accessing Data from Multiple Members

An additional advantage of the QCMD command is that it allows you to change dynamically the physical file member against which you are working. To switch dynamically between file members, use the OS/400 Override Database File (OVRDBF) command. To execute this command remotely from a Progress client, use QCMD.

For example, assume that the Progress PRODEMO database STATE file has two members: one named STATE, the other EUROPE. The member named STATE contains the U.S. state information, and the EUROPE member holds similar information for European countries.

Suppose, for example, that you start a session as normal and execute this procedure:

```
FOR EACH state:
DISPLAY state.
```

The output lists all of the U.S. states in the database file STATE, member STATE. To then run the same procedure but get a list of the European states, follow these steps:

1 ♦ Enter this QCMD command to close the STATE file immediately:

```
CREATE QCMD.
ASSIGN cmd = "!CLOSE STATE".
RELEASE.
```

2 ♦ Enter this QCMD command to access the data in the STATE, file EUROPE member:

```
CREATE QCMD.
ASSIGN cmd = "! OVRDBF FILE(STATE) MBR(EUROPE)".
RELEASE.
```

3 ♦ Rerun the original Progress procedure:

```
FOR EACH state:
DISPLAY state.
```

The output will be the data from the EUROPE member. Use this technique to access data from multiple members.

NOTE: Members must have the same data format.

11.2.5 SBMJOB Parameters

This section lists the parameters for the OS/400 Submit Job (SBMJOB) CL command. It also lists the Progress equivalents (what Progress displays when you access the parameter form with the INSERT statement) and the OS/400 default for that parameter. See the *AS/400 CL Reference* for more information about the SBMJOB command. Table 11–4 describes the SBMJOB parameters.

Table 11–4: SBMJOB Parameters *(1 of 3)*

Keyword	Parameter	Value
CMD	Command to run	Names the OS/400 CL command to submit.
JOB	Job name	Names the job that you are submitting. The default is *JOBD.
JOBD	Job description	Names the job description. The default is *USRPRF.
	Library	Names the library of the job description.
JOBQ	Job queue	Names the job queue. The default is *JOBD.
	Library	Names the job queue's library.

Table 11–4: SBMJOB Parameters *(2 of 3)*

Keyword	Parameter	Value
JOBPTY	Job priority on JOBQ	Names the priority on the job queue. The default is *JOBD.
OUTPTY	Output priority on OUTQ	Names the priority for output files that the job creates. The default is *JOBD.
PRTDEV	Print device	Names the printer. The default is *CURRENT.
OUTQ	Output queue	Names the output queue for the job's spooled output. The default is *CURRENT.
	Library	Names the output queue's library.
USER	User	Names the user profile. The default is *CURRENT.
PRTTXT	Print text	Identifies the text printed at the bottom of this job's printed output. The default is *CURRENT.
RTGDTA	Routing data	Names the routing data that starts the first routing step. The default is QCMDB.
RQSDTA	Request data or command	Names the request data used as the last entry in the message queue. The default is *CMD.
SYSLIBL	System library list	Names the system library list that the job should use. The default is *CURRENT.
CURLIB	Current library	Names the current library that the job should use. The default is *CURRENT.
INLLIB	Initial library list	Names the initial library list that the job should use. (This is the user portion of the library list.) The default is *CURRENT.
LOG[1]	Message logging level severity text	Names the values that the job uses to write messages to the job log. You can specify message level, severity, and text. The default is *JOBD.
LOGCLPGM	Log CL program commands	Indicates whether to write CL commands to the job log. The default is *JOBD.

Table 11–4: SBMJOB Parameters *(3 of 3)*

Keyword	Parameter	Value
INQMSGRPY	Inquiry message reply	Identifies how to answer inquiry messages. The default is *JOBD.
HOLD	Hold on job queue	Identifies whether the job is held when put on the job queue. The default is *JOBD.
DATE	Job date	Names the date that the job is started. The default is *JOBD.
SWS	Job switches	Names the job switches that the job uses. The default is *JOBD.
DSPSMBJOB	Allow display by WRKSBMJOB	Identifies whether to display the job on the screen when the WRKSBMJOB command is entered. The default is *YES.
MSGQ	Message queue	Names the message queue where messages are written when the job completes (either successfully or with errors). The default is *USRPRF.
	Library	Names the library where the message queue resides.
CCISD	Coded character set ID	Names the coded character set ID to use with this job. The default is *CURRENT.

[1] The field name in the QCMD file is msglog.

11.3 Progress/400 Stored Procedure Support

The Progress/400 Product allows you to call OS/400 programs from within a Progress 4GL procedure.

A stored procedure as it pertains to DB2/400 and Progress/400 is any program object (Type = *PGM) created with any of the available languages on OS/400: CL, RPG/400, ILE RPG/400, COBOL/400, ILE COBOL/400, ILE C/400, PL1/400, FORTRAN/400, or REXX. A stored procedures within a Progress 4GL-based application can enhance performance of batch-style functions. Any sort of a bulk database manipulation or specialized calculation that takes an excessive amount of time is a candidate for a program to run as a stored procedure.

The Progress/400 Data Dictionary allows you to define and maintain procedures and their parameters. Procedure definitions are stored in the P__FILE file, and parameter definitions are stored in the P__FIELD file. When Progress synchronizes the schema holder, the schema image maintains this information in the _file and _field files marked as type AS/400 Procedures. See Chapter 9, "Remote Client DB2/400 Utilities," for information about maintaining stored procedures.

Table 11–5 lists information contained in a stored procedure.

Table 11–5: Information Contained in a Stored Procedure

Contents	Description
Stored Procedure Name	Progress/400 stored procedure name. Use this name in 4GL code. The name field supports up to 30 characters.
AS/400 Program and Library Name	The name of the AS/400 program object to execute. This is limited to 10 characters.
PROCEDURE	Identifies a _file schema entry as a stored procedure definition and not a database file in the schema holder.
Parameter Name	Defines a formal name of a parameter for the stored procedure. This name can be used in the 4GL RUN STORED-PROC statement.
Parameter Position	Specifies the position of a specific parameter relative to other parameters used by the stored procedure.
Input Parameter Output Parameter Input-Output Parameter	Defines a stored procedure parameter as: Input: provided by the 4GL application Output: returned to the 4GL application Both: Input and Output
Storage Length	The length of a field in storage is mapped to fixed values for certain data types or based on the format of others.

You run stored procedures from within Progress procedures by using the 4GL RUN STORED-PROC statement. The OS/400 procedure that you run with this statement can return two types of values:

- Return codes that indicate whether the stored procedure succeeded
- Values of output parameters that you define when you create the procedures

If a stored procedure fails a run time, you will receive the error "Unable to Open File." Press **ENTER** and a dialog box appears with the specific AS/400 message that caused the stored procedure to stop.

Stored procedures called from within Progress applications cannot return Dates or RECID data types. Table 11–6 lists the supported data types.

Table 11–6: Stored Procedure Argument Data Types

OS/400 Data Type	Progress Data Type
Zoned Numeric Packed Decimal Pckd Decimal (even digits)	Progress represents all three data types by the Decimal data type in the schema holder.
Character Alpha (fixed length)	Character data type in the schema holder.
Short Integer Long Integer	The Progress Integer data type in the schema holder represents both data types.
Logical	Progress Logical data type.

A maximum of 32 parameters can be defined for each procedure. Each procedure must have a unique name. Therefore, you cannot redefine a procedure with different sets of parameters without duplicating the program in the listed library.

The Progress 4GL has statements and functions that allow you to use the return codes and values of the output parameters. Table 11–7 lists these statements and functions.

Table 11–7: Functions for Stored Procedure Return Codes

Progress 4GL	Description
CLOSE STORED-PROC statement	Retrieves the values from the output parameters you defined for the stored procedure and tells Progress that the stored procedure has ended.
PROC-HANDLE function	Allows you to specify a handle to identify a stored procedure.
PROC-STATUS function	Reads the return value.

The following sections describe how to run Progress/400 stored procedures and retrieve return codes and output parameter values.

11.3.1 Running a Stored Procedure

The Progress 4GL statement RUN STORED-PROC allows you to run a Progress/400 stored procedure. You must indicate the end of a stored procedure in your progress procedure by using the CLOSE STORED-PROC statement.

This is the partial syntax for the RUN STORED-PROC statement:

SYNTAX

```
RUN STORED-PROCprocedure [NO-ERROR] [([input] parameter,
     [output] parameter [input-output] parameter]
```

This is the partial syntax for the CLOSED STORED-PROC statement:

SYNTAX

```
CLOSE STORED-PROC procedure
```

For example, the following Progress 4GL code runs a Progress/400 stored procedure `pcust`:

```
DEFINE VARIABLE sts  AS INTEGER.
DEFINE VARIABLE h1   AS INTEGER.

RUN STORED-PROC pcust h1 = PROC-HANDLE (input “SERVE”, input 5.5).
CLOSE STORED-PROC my-procedure sts = PROC-STATUS.
IF sts = 0 THEN DO:
...
END.
ELSE DO:
...
END.
```

This code defines an integer variable that serves as a handle for identifying the stored procedure and a variable to hold the procedure status return value. If you have only one active stored procedure, you do not have to specify a handle. However, it is good programming practice to use handles to identify all your stored procedures.

The stored procedure passes the values "SERVE" and 5.5 to the two parameters that have been defined in the schema as input parameters. The Progress procedures uses the CLOSE STORED-PROC statement to notify the client that the procedure is ended and that the PROC-STATUS can be interrogated.

The stored procedure return codes provide information on its status. These codes indicate whether the stored procedure succeeded. The above code evaluates the `sts` variable to determine which block of code to execute.

You can close all stored procedures at once with the following statement:

```
RUN STORED_PROC closeallprocs.
```

Retrieving Output Parameter Values

When you call a stored procedure you can specify the ordered list of positional parameters or you can name the parameters individually. To retrieve output parameter values from a stored procedure, you request them with the keyword OUTPUT or INPUT-OUTPUT. You must specify the parameters in the 4GL procedure exactly as they were defined in the server schema.

Programming Restrictions

Table 11–8 lists restrictions when implementing Progress/400 Stored Procedures.

Table 11–8: Stored Procedure Implementation Restrictions

Programming Consideration	Description
4GL Transactions	An AS/400 stored procedure will not be included in a 4GL-managed transaction. The stored procedure program runs independently from the DataServer and performs its tasks within its own transaction scope, that is commitment control. Specifically, if the 4GL transaction is rolled back (UNDO, LEAVE) after the execution of a stored procedure, the operations performed by that program cannot be undone.
Verification of procedure parameters	It is not possible to interrogate an OS/400 program object to determine its parameters (both number and data type) in all cases. Therefore, verification of procedure parameters (that must be manually entered in the client Progress/400 dictionary) is not possible. Progress/400 determines incorrect parameters at execution time only.
Data types for procedure parameters	The list of data types applicable to stored procedures is a subset of the database field types. However, that subset differs based on the language used to create the stored procedure program. Therefore, Progress/400 uses the complete list of data types as defined for database files.

11.4 OS/400 Object Access

The Progress/400 product supports Progress 4GL access to data areas, user spaces, and data queues. AS/400 applications and IBM use these objects and their respective APIs to facilitate interprocess communication, storage of control information, and support of the AS/400 APIs. A Progress-based 4GL application written for the AS/400 can now utilize the same functionality as RPG or COBOL programs.

A Progress 4GL program running on a client or natively on the AS/400 can access these objects through a consistent set of 4GL APIs. A 4GL program that utilizes these APIs can be written for an n-tier configuration, and it will also run on the AS/400 using the Progress/400 Native 4GL Client.

11.4.1 Data Area Support

Progress/400 DataServer allows you to perform operations on data areas from within a Progress client session. You can perform two types of operations:

- Change data in a data area using the Progress/400 CHANGE DATA AREA (`chgdara.p`) API. Note that you can send only character data to a data area.
- Retrieve information from a data area using the Progress/400 RETRIEVE DATA AREA (`rtvdara.p`) API. Note that you can retrieve only character data from a data area.

Using these APIs insulates your code from possible changes to the underlying method of changing and retrieving data from data areas and provides a standard interface that does not change. You use them in the same way regardless of whether your code is being run from the remote client or from the native clients.

Progress/400 supports the change and retrieve operations by providing the QDTAARA table definition in the Progress/400 schema. This table is similar to the QCMD table in that it controls the function of Progress/400, but different in that data can be sent to and received from data areas. More than one job can place data onto or receive data from a particular data area.

NOTE: Do not use the QDTAARA name as a physical filename: the SYNC function updates the file on the client, making the data area APIs unusable.

Before you can place information into a data area, you must create the data area using the OS/400 Create Data Area (CRTDTAARA) command. To display the data area, use the OS/400 Display Data Area (DSPDTAARA) command. For detailed information about OS/400 data area access routines, see the IBM OS/400 *Control Language Programmer's Guide*.

Progress displays error messages to the remote client session, with more detailed error information available in the OS/400 job log (assuming that the server is started with logging), or in the native client's job log, as appropriate.

NOTE: Progress/400 does not support accessing the following two types of data areas: *GDA (Group Data Area), *PDA (Pre-start Data Area).

CHANGE DATA AREA (chgdara.p) API

Table 11–9 describes the parameters that you must pass to the CHANGE DATA AREA API. You can use the same variable names or names of your own choice. Note that you must pass **all** of these parameters to the API, and they must be in the same order as in Table 11–9.

Table 11–9: CHANGE DATA AREA (chgdara.p) Parameters

Parameter Name	Parameter Type and Data Type	Value
db-name	INPUT CHARACTER	The name of a connected DB2/400 database that contains the QDTAARA table. The database must be connected before the call to the API because an alias will be defined for the database that allows any DB2/400 database to use the API.
area-name[1]	INPUT CHARACTER	The name of the data area on the AS/400. This data area must have been created before using the API.
library-name	INPUT CHARACTER	The library name where the data area resides.
start-position	INPUT INTEGER	The starting position of the data to be changed in the data area. The starting position is 1 based, so the first position in the data area is 1.
data-length	INPUT INTEGER	The length of the data to be changed in the data area.
data-value	INPUT/OUTPUT CHARACTER	The data to be placed into the data area. A data area can support a character string as long as 2000 characters, 1024 for *LDA.
status	OUTPUT INTEGER	A return parameter that indicates whether the entry has been changed.

[1] You can use *LDA for area-name. When you use *LDA, you must blank out library-name. The *LDA value specifies that the contents of the local data area are changed. The local data area is automatically associated with your job on the AS/400 by the operating system (OS/400) and is 1024 characters long. For more detailed information on the use of *LDA, see the "Local Data Area" section in the *CL Programmer's Guide* of the IBM documentation.

The following example illustrates calling the CHANGE DATA AREA API from your code:

```
RUN as4dict/chgdara.p (INPUT db-name, INPUT area-name,
                       INPUT library-name, INPUT start-pos,
                       INPUT data-length, INPUT/OUTPUT data-value,
                       OUTPUT status),
```

To verify that an entry has been changed, check the value of the OUTPUT parameter. Table 11–10 lists possible return values.

Table 11–10: CHANGE DATA AREA (chgdara.p) Return Values

Status	Reason	Client
0	The entry has been placed in the data area.	Remote and native
-1	Either the DB2/400 database name is incorrect or the database is not connected.	Remote only
-2	The named DB2/400 database does not contain the QDTAARA table.	Remote only
-3	The start position must be greater than zero.	Remote and native
-4	The data length must be less than 2000.	Remote and native
-5	The start position plus the data length minus 1 must be less than or equal to 2000.	Remote and native
-6	A general error occurred.	Native only

RETRIEVE DATA AREA (rtvdara.p) API

Table 11–11 describes the parameters that you must pass to the RETRIEVE DATA AREA API. You can use the same variable names or names of your own choice. Note that you must pass **all** of these parameters to the API, and they must be in the same order as in Table 11–11.

Table 11–11: RETRIEVE DATA AREA (rtvdara.p) Parameters

Parameter Name	Parameter Type and Data Type	Value
db-name	INPUT CHARACTER	The name of a connected DB2/400 database that contains the QDTAARA table. The database must be connected before the call to the API because an alias will be defined for the database that allows any DB2/400 database to use the API.
area-name[1]	INPUT CHARACTER	The name of the data area on the AS/400. This data area must have been created before using the API.
library-name	INPUT CHARACTER	The library name where the data area resides
start-position	INPUT INTEGER	The starting position of the data to be retrieved from the data area. The starting position is 1 based, so the first position in the data area is 1.
data-length	INPUT INTEGER	The length of the data to be retrieved from the data area.
data-value	INPUT/OUTPUT CHARACTER	The data to be retrieved from the data area. A data area can support a character string as long as 2000 characters, 1024 for *LDA.
status	OUTPUT INTEGER	A return parameter that indicates whether the entry has been retrieved.

[1] You can use *LDA for area-name. When you use *LDA, you must blank out library-name. The *LDA value specifies that the contents of the local data area are changed. The local data area is automatically associated with your job on the AS/400 by the operating system (OS/400) and is 1024 characters long. For more detailed information on the use of *LDA, see the "Local Data Area" section of the *CL Programmer's Guide* in the IBM documentation.

The following example illustrates calling the RETRIEVE DATA AREA API from your code:

```
RUN as4dict/rtvdara.p (INPUT db-name, INPUT area-name,
                       INPUT library-name, INPUT start-pos,
                       INPUT data-length, INPUT/OUTPUT data-value,
                       OUTPUT status),
```

To verify that an entry has been retrieved, check the value of the OUTPUT parameter. Table 11–12 lists possible return values.

Table 11–12: RETRIEVE DATA AREA (rtvdara.p) Return Values

Status	Reason	Client
0	The entry has been retrieved from the data area.	Remote and native
-1	Either the DB2/400 database name is incorrect or the database is not connected.	Remote only
-2	The named DB2/400 database does not contain the QDTAARA table.	Remote only
-3	The start position must be greater than zero.	Remote and native
-4	The data length must be less than 2000.	Remote and native
-5	The start position plus the data length minus 1 must be less than or equal to 2000.	Remote and native
-6	A general error occurred.	Native only

Data Area Coding Example

The following example code illustrates how to change and retrieve entries from a data area:

```
DEFINE VARIABLE db-name AS CHARACTER NO-UNDO.
DEFINE VARIABLE area-name AS CHARACTER NO-UNDO.
DEFINE VARIABLE lib-name AS CHARACTER NO-UNDO.
DEFINE VARIABLE data-length AS INTEGER NO-UNDO.
DEFINE VARIABLE data-value AS CHARACTER NO-UNDO.
DEFINE VARIABLE stat AS INTEGER NO-UNDO.

ASSIGN db-name = "mydb"
       area-name = "mydataarea"
       lib-name = "mydb"
       data-value = "Changing data in data area"
       data-length = LENGTH(data-value).


/* An example of sending an entry to a data area */
RUN as4dict/chgdara.p (INPUT db-name, INPUT area-name,
                       INPUT lib-name, INPUT data-length,
                       Input-Output data-value,
                       OUTPUT stat).

/* If the entry was changed in the data area,          */
/* the value of stat will equal the length of the data */
IF stat = LENGTH(data-value) THEN DO:
   ASSIGN data-value = "".

/* An example of retrieving an entry from a data area */
RUN as4dict/rtvdara.p (INPUT db-name, INPUT area-name,
                       INPUT lib-name, INPUT data-length,
                       Input-Output data-value,
                       OUTPUT stat).

IF stat > 0 THEN
   DISPLAY data-value WITH FRAME entry NO-LABELS.
ELSE
   DISPLAY stat WITH FRAME status NO-LABELS.

END
```

11.4.2 User Space Support

A user space is a permanent object that consists of a collection of bytes used for storing user-defined information. Progress/400 uses user spaces to store large amounts of data and pass this data from job to job or from system to system. Progress/400 allows you to perform operations on user spaces from within a Progress client session. You can perform two types of operations:

- Change data in a user space, using the Progress/400 CHANGE USER SPACE (`chguspc.p`) API.
- Retrieve information from a user space, using the Progress/400 RETRIEVE USER SPACE (`rtvuspc.p`) API.

Using these APIs insulates your code from possible changes to the underlying method of changing and retrieving data from user spaces and provides a standard interface that does not change. You use them in the same way regardless of whether the code is being run from the remote client or from the native clients.

Progress/400 supports the change and retrieve operations by providing the QUSRSPC table definition in the Progress/400 schema. This table is similar to the QCMD table in that it controls the function of the Progress/400 DataServer, but different in that information can be changed in and retrieved from QUSRSPC.

NOTE: Do not use the QUSRSPC name as a physical file name: the SYNC function updates the file on the client, making the user space APIs unusable.

Note that if you retrieve numeric data or pointers stored in the user space, you must use the SUBSTRING function on the contents of the returned buffer in order to display the data. This is because if Progress encounters a null within the returned numeric or pointer data, it interprets this as the end of the buffer and no further information is displayed. For additional information on using the SUBSTRING function, see the "Using the LOOKUP and SUBSTRING Functions with Send Data" section.

Before you can place information into a user space, you must create the user space using the OS/400 Create User Space (QUSCRTUS) command. For detailed information about OS/400 user space access routines, see the IBM OS/400 *System API Reference*.

Progress displays error messages in the Progress client session, and more detailed information might be available in the OS/400 job log.

CHANGE USER SPACE (chguspc.p) API

Table 11–13 describes the parameters that you must pass to the CHANGE USER SPACE API. You can use the same variable names or names of your own choice. Note that you must pass **all** of these parameters to the API, and they must be in the same order as in Table 11–13.

Table 11–13: CHANGE USER SPACE (chguspc.p) Parameters

Parameter Name	Parameter Type and Data Type	Value
db-name	INPUT CHARACTER	The name of a connected DB2/400 database that contains the QUSRSPC table. The database must be connected before the call to the API because an alias will be defined for the database that allows any DB2/400 database to use the API.
uspc-name	INPUT CHARACTER	The name of the user space on the AS/400. This user space must have been created before using the API.
library-name	INPUT CHARACTER	The library name where the user space resides.
start-position	INPUT INTEGER	The starting position of the data to be changed in the user space. The starting position is 1 based, so the first position in the user space is 1.
data-length	INPUT INTEGER	The length of the data to be changed in the user space.
data-value	INPUT/OUTPUT CHARACTER	The data to be placed into the user space. A user space can support up to 16 megabytes of information; however, this utility allows you to access only 2048 bytes at one time.
status	OUTPUT INTEGER	A return parameter that indicates whether the entry has been changed.

The following example illustrates calling the CHANGE USER SPACE API from your code:

```
RUN as4dict/chguspc.p (INPUT db-name, INPUT uspc-name,
                       INPUT library-name, INPUT start-pos,
                      INPUT data-length, INPUT/OUTPUT data-value,
                       OUTPUT status),
```

To verify that an entry has been changed, check the value of the OUTPUT parameter. Table 11–14 lists possible return values.

Table 11–14: CHANGE USER SPACE (chguspc.p) Return Values

Status	Reason	Client
#	The length of the data entry. This number is either 0, which means the entry was not changed, or the length of the data.	Remote and native
-1	Either the DB2/400 database name is incorrect or the database is not connected.	Remote only
-2	The named DB2/400 database does not contain the QUSRSPC table.	Remote only
-3	The start position must be greater than zero.	Remote and native
-4	The data length must be less than 2048.	Remote and native

RETRIEVE USER SPACE (rtvuspc.p) API

Table 11–15 describes the parameters that you must pass to the RETRIEVE USER SPACE API. You can use the same variable names or names of your own choice. Note that you must pass **all** of these parameters to the API, and they must be in the same order as in Table 11–15.

Table 11–15: RETRIEVE USER SPACE (rtvuspc.p) Parameters

Parameter Name	Parameter Type and Data Type	Value
db-name	INPUT CHARACTER	The name of a connected DB2/400 database that contains the QUSRSPC table. The database must be connected before the call to the API because an alias will be defined for the database that allows any DB2/400 database to use the API.
uspc-name	INPUT CHARACTER	The name of the user space on the AS/400. This user space must have been created before using the API.
library-name	INPUT CHARACTER	The library name where the user space resides.
start-position	INPUT INTEGER	The starting position of the data to be retrieved from the user space. The starting position is 1 based, so the first position in the data area is 1.
data-length	INPUT INTEGER	The length of the data to be changed in the user space.
data-value	INPUT/OUTPUT CHARACTER	The data to be retrieved from the user space. A user space can support up to 16 megabytes of information; however, this utility allows you to access only 2048 bytes at one time.
status	OUTPUT INTEGER	A return parameter that indicates whether the entry has been retrieved.

The following example illustrates calling the RETRIEVE USER SPACE API from your code:

```
RUN as4dict/rtvuspc.p (INPUT db-name, INPUT uspc-name,
                       INPUT library-name, INPUT start-pos,
                       INPUT data-length, INPUT/OUTPUT data-value,
                       OUTPUT status),
```

To verify that an entry has been retrieved, check the value of the OUTPUT parameter. Table 11–16 lists possible return values.

Table 11–16: RETRIEVE USER SPACE (rtvuspc.p) Return Values

Status	Reason	Client
#	The length of the data entry. This number is either 0, which means the entry was not changed, or the length of the data.	Remote and native
-1	Either the DB2/400 database name is incorrect or the database is not connected.	Remote only
-2	The named DB2/400 database does not contain the QUSRSPC table.	Remote only
-3	The start position must be greater than zero.	Remote and native
-4	The data length must be less than 2048.	Remote and native

User Space Coding Example

The following example code illustrates how to change and retrieve entries from a user space:

```
DEFINE VARIABLE db-name AS CHARACTER NO-UNDO.
DEFINE VARIABLE uspc-name AS CHARACTER NO-UNDO.
DEFINE VARIABLE lib-name AS CHARACTER NO-UNDO.
DEFINE VARIABLE data-length AS INTEGER NO-UNDO.
DEFINE VARIABLE data-value AS CHARACTER NO-UNDO.
DEFINE VARIABLE stat AS INTEGER NO-UNDO.

ASSIGN db-name = "mydb"
       uspc-name = "myuserspc"
       lib-name = "mydb"
       data-value = "Changing data in user space"
       data-length = LENGTH(data-value).

/* An example of sending an entry to a data area */
RUN as4dict/chguspc.p (INPUT db-name, INPUT uspc-name,
                       INPUT lib-name, INPUT data-length,
                       Input-Output data-value,
                       OUTPUT stat).

/* If the entry was changed in the user space,         */
/* the value of stat will equal the length of the data */
IF stat = LENGTH(data-value) THEN DO:
   ASSIGN data-value = "".

/* An example of retrieving an entry from a user space */
RUN as4dict/rtvuspc.p (INPUT db-name, INPUT uspc-name,
                       INPUT lib-name, INPUT data-length,
                       Input-Output data-value,
                       OUTPUT stat).

IF stat = LENGTH(data-length) THEN
   DISPLAY data-value WITH FRAME entry NO-LABELS.
ELSE
   DISPLAY stat WITH FRAME status NO-LABELS.
END.
```

11.4.3 Data Queue Support

Progress/400 allows you to perform operations on data queues from within a Progress client session. You can perform two types of operations:

- Create a record and send it to a data queue, using the Progress/400 SEND DATA QUEUE (`snddtaqe.p`) API.
- Find and receive a record from the data queue, using the Progress/400 RECEIVE DATA QUEUE (`rcvdtaqe.p`) API.

Using these APIs insulates your code from possible changes to the underlying method of sending and receiving data from data queues and provides a standard interface that will not change. You use them in the same way regardless of whether the code runs from the remote client or from the native clients.

Progress/400 supports the send (create) and receive (find) operations by providing the QDTAQ-ENTRY table definition in the Progress/400 schema. This table is similar to the QCMD table in that it controls the function of the Progress/400 DataServer, but different in that data can be sent to and received from data queues. More than one job can place data onto or receive data from a particular data queue.

NOTE: Do not use the QDTAQ-ENTRY name as a physical file name because the SYNC function updates the file on the client, making the data queue APIs unusable.

Before you can place a record onto a data queue, you must create the data queue using the OS/400 Create Data Queue (CRTDTAQ) command. For detailed information about OS/400 data queue access routines, see the IBM OS/400 *Control Language Programmer's Guide* and the *System API Reference*.

The DataServer displays appropriate error messages to the remote client session, with more detailed error information available in the OS/400 job log (assuming that the server is started with logging), or to the native client's job log, as appropriate.

SEND DATA QUEUE (snddtaqe.p) API

Table 11–17 describes the parameters that you must pass to the SEND DATA QUEUE API. You can use either the same variable names or names of your own choice. Note that you must pass **all** of these parameters to the API, and they must be in the same order as in Table 11–17.

Table 11–17: SEND DATA QUEUE (snddtaqe.p) Parameters *(1 of 2)*

Parameter Name	Parameter Type and Data Type	Value
db-name	INPUT CHARACTER	The name of a connected DB2/400 database that contains the QDTAQ-ENTRY table. The database must be connected before the call to the API because an alias will be defined for the database that allows any DB2/400 database to use the API.
que-name	INPUT CHARACTER	The name of the data queue on the AS/400. This data queue must have been created before using the API.

Table 11–17: SEND DATA QUEUE (snddtaqe.p) Parameters *(2 of 2)*

Parameter Name	Parameter Type and Data Type	Value
library-name	INPUT CHARACTER	The library name where the data queue resides.
entry-data	INPUT CHARACTER	The data to be placed on the data queue.
key-data	INPUT CHARACTER	If the data queue was created to use a key, the data for the key. If the data queue is not keyed, assign this parameter to blank.
key-length	INPUT INTEGER	If the data queue was created to use a key, the length of the key specified when the data queue was created. If the data queue is not keyed, assign 0 to the parameter.
Status	OUTPUT INTEGER	A return parameter that indicates whether the entry has been placed on the data queue.

The following example illustrates calling the SEND DATA QUEUE API from your code:

```
RUN as4dict/snddtaqe.p (INPUT db-name, INPUT que-name,
                        INPUT library-name, INPUT entry-data,
                        INPUT key-data, INPUT key-length,
                        OUTPUT status),
```

The OUTPUT parameter can be checked to verify that the entry has been placed on the queue.

To verify that an entry has been changed, check the value of the OUTPUT parameter. Table 11–18 lists possible return values.

Table 11–18: SEND DATA QUEUE (snddtaqe.p) Return Values

Status	Reason	Client
0	The entry has been placed onto the data queue.	Remote and native
-1	Either the DB2/400 database name is incorrect or the database is not connected.	Remote only
-2	The named DB2/400 database does not contain the QDTAQ-ENTRY table.	Remote only
-4	An error occurred and an entry was not placed on the data queue.	Remote and native

11.4.4 Using the LOOKUP and SUBSTRING Functions with Send Data

When using the SUBSTRING function to process send data in conjunction with a LOOKUP function, **do not** use the following syntax:

```
IF LOOKUP(SUBSTRING(send-data,1,1),entry-list) = 0 THEN your-code
```

The entry-list variable contains a list of values known to be in the first position, but this syntax fails. If you reassign the send-data variable to another CHARACTER variable within the same procedure as the LOOKUP function, the syntax works correctly.

If you use the SUBSTRING function without doing a LOOKUP, as shown in the following syntax, the SUBSTRING function works correctly:

```
IF SUBSTRING(send-data,1,1) = "A" THEN your-code
```

RECEIVE DATA QUEUE (rcvdtaqe.p) API

Table 11–19 describes the parameters that you must pass to the RECEIVE DATA QUEUE API. You can use the same variable names or names of your own choice. Note that you must pass **all** of these parameters to the API, and they must be in the same order as in Table 11–19.

Table 11–19: RECEIVE DATA QUEUE (rcvdtaqe.p) Parameters *(1 of 2)*

Parameter Name	Parameter Type and Data Type	Value
db-name	INPUT CHARACTER	The name of a connected DB2/400 database that contains the QDTAQ-ENTRY table. The database must be connected before the call to the API because an alias will be defined for the database that allows any DB2/400 database to use the API.
que-name	INPUT CHARACTER	The name of the data queue on the AS/400. This data queue must have been created before using the API.
library-name	INPUT CHARACTER	The library name where the data queue resides.
key-data	INPUT CHARACTER	If the data queue was created to use a key, the data for the key. If the data queue is not keyed, assign this parameter to blank.
key-length	INPUT INTEGER	If the data queue was created to use a key, the length of the key specified when the data queue was created. If the data queue is not keyed, assign 0 to this parameter.
key-order	INPUT CHARACTER	The operand for the key data. The parameter can contain one of the following values: EQ, GE, GT, LE, LT, NE, =, >=, <, <=, <, <>. If the data queue is not keyed, assign blanks to this parameter.

Table 11–19: RECEIVE DATA QUEUE (rcvdtaqe.p) Parameters *(2 of 2)*

wait-time	INPUT INTEGER	The amount of time, in seconds to wait for an entry. If a negative number is specified, the API waits until an entry is available. If zero is specified, the API returns immediately. A positive number specifies the number of seconds that the API should wait.
sender-Data	INPUT/OUTPUT CHARACTER	Notifies the API whether the sender data is wanted and should be returned. Assign Y to the variable if you want data and N or blank if you do not. The maximum length available is 64 bytes but the maximum length returned is 56: the first eight bytes are not valid to Progress and are removed before being returned. See the IBM System API Reference for an explanation of the first eight bytes.
entry-data	OUTPUT CHARACTER	The entry removed from the data queue. You must know what the data will contain in order to process it with the SUBSTRING function for use in your procedure. Be careful about having the hex character zero in the data: Progress interprets the hex character zero as a NULL and the end of the data. For additional information on using the SUBSTRING function, see the "Using the LOOKUP and SUBSTRING Functions with Send Data" section.
Status	OUTPUT INTEGER	A return parameter that indicates whether the entry has been placed on the data queue.

The following example illustrates calling the RECEIVE DATA QUEUE API from your code:

```
RUN as4dict/rcvdtaqe.p (INPUT db-name, INPUT que-name,
                        INPUT library-name, INPUT key-data,
                        INPUT key-length, INPUT key-order,
                        INPUT wait-time, INPUT/OUTPUT sender-data,
                        OUTPUT entry-data, OUTPUT status),
```

To verify that an entry has been received, check the value of the OUTPUT parameter. Table 11–20 lists possible return values.

Table 11–20: RECEIVE DATA QUEUE (rcvdtaqe.p) Return Values

Status	Reason	Client
-1	Either the DB2/400 database name is incorrect or the database is not connected.	Remote only
-2	The named DB2/400 database does not contain the QDTAQ-ENTRY table.	Remote only
-3	The operand in the key-order parameter did not contain a correct value.	Remote and native
#	The length of the data entry. This number can be any value from 0, which means that no entry was returned, to the length of the data.	Remote and native

Data Queue Coding Examples

The examples in this section illustrate how to send and receive entries from unkeyed and keyed data queues.

The first example sends and receives an entry from an **unkeyed** data queue:

```
DEFINE VARIABLE db-name   AS CHARACTER NO-UNDO.
DEFINE VARIABLE que-name  AS CHARACTER NO-UNDO.
DEFINE VARIABLE lib-name  AS CHARACTER NO-UNDO.
DEFINE VARIABLE ent-data  AS CHARACTER NO-UNDO.
DEFINE VARIABLE ky-data   AS CHARACTER NO-UNDO.
DEFINE VARIABLE ky-length AS INTEGER  NO-UNDO.
DEFINE VARIABLE wait-time AS INTEGER  NO-UNDO.
DEFINE VARIABLE key-order AS CHARACTER NO-UNDO.
DEFINE VARIABLE stat      AS INTEGER   NO-UNDO.
DEFINE VARIABLE send-data AS CHARACTER NO-UNDO.

ASSIGN db-name   = "mydb"
       que-name  = "dtaqnokey"
       lib-name = "mydb"
       ent-data  = "Sending data to unkeyed data queue"
       ky-data   = ""
       key-order = ""
       ky-length = 0.

/* An example of sending an entry to a data queue */
RUN as4dict/snddtaqe.p (INPUT db-name, INPUT que-name,
                        INPUT lib-name, INPUT ent-data,
                        INPUT ky-data,  INPUT ky-length,
                        OUTPUT stat).

/* Entry is placed on queue if stat variable equals 0 */
IF stat = 0 THEN DO:
   ASSIGN wait-time = 0
   send-data = "Y".

/*An example of requesting an entry from a data queue */
RUN as4dict/rcvdtaqe.p (INPUT db-name, INPUT que-name,
                        INPUT lib-name, INPUT ky-data,
                        INPUT ky-length, INPUT key-order,
                        INPUT wait-time INPUT-OUTPUT send-data,
                        OUTPUT ent-data, OUTPUT stat).

/* stat will be the length of the entry received or 0 if not received. */
IF STAT > 0 THEN
   DISPLAY ent-data FORMAT "x(40)" send-data format "x(56)"
   WITH FRAME entry NO-LABELS.
ELSE
   DISPLAY stat WITH FRAME status NO-LABELS.
END.
```

This example sends and receives an entry from a **keyed** data queue. This data queue was created with a key length of 10 and a request for sender information:

```
DEFINE VARIABLE db-name   AS CHARACTER NO-UNDO.
DEFINE VARIABLE que-name  AS CHARACTER NO-UNDO.
DEFINE VARIABLE lib-name  AS CHARACTER NO-UNDO.
DEFINE VARIABLE ent-data  AS CHARACTER NO-UNDO.
DEFINE VARIABLE ky-data   AS CHARACTER NO-UNDO.
DEFINE VARIABLE ky-length AS INTEGER NO-UNDO.
DEFINE VARIABLE wait-time AS INTEGER NO-UNDO.
DEFINE VARIABLE key-order AS CHARACTER NO-UNDO.
DEFINE VARIABLE stat      AS INTEGER   NO-UNDO.
DEFINE VARIABLE send-data AS CHARACTER NO-UNDO.

ASSIGN db-name    = "mydb"
       que-name  = "keyedqueue"
       lib-name  = "mydb"
       ent-data  = "Sending data to a keyed queue"
       ky-data   = "myname"
       key-order = "EQ"
       ky-length = 10.

/* An example of sending an entry to a data queue */
RUN as4dict/snddtaqe.p (INPUT db-name, INPUT que-name,
                        INPUT lib-name, INPUT ent-data,
                        INPUT ky-data, INPUT ky-length,
                        OUTPUT stat).

/* Entry is placed on queue if stat variable equals 0 */
IF stat = 0 THEN DO:
   ASSIGN wait-time = 0
   send-data = "Y".

/*An example of requesting an entry from a data queue */
RUN as4dict/rcvdtaqe.p (INPUT db-name, INPUT que-name,
                        INPUT lib-name, INPUT ky-data,
                        INPUT ky-length, INPUT key-order,
                        INPUT wait-time INPUT-OUTPUT send-data,
                        OUTPUT ent-data, OUTPUT stat).

/* stat will be the length of the entry received or 0 if not received. */
IF STAT > 0 THEN
   DISPLAY ent-data FORMAT "x(40)" send-data format "x(56)".
ELSE
   DISPLAY stat.
END.
```

A

Tutorials for Managing Your Dictionary Library

This appendix documents how to use the current version of Progress/400 to manage your Dictionary Library. Specifically, this appendix addresses:

- Creating a new DB2/400 database with Progress/400
- Creating a copy of a server schema and data
- Modifying existing DB2/400 data definitions with Progress/400

A.1 Creating a New DB2/400 Database with Progress/400

This section explains how to create a DB2/400 database when you are not converting from an earlier Progress/400 version. It describes two options:

- Creating a new database with no existing DB2/400 data files
- Creating a database that accesses existing DB2/400 data files; for example, that used by RPG programs

Each set of instructions also includes creating the required server schema.

Note that you can create a database that contains both new **and** existing data files. In this case, follow the steps for creating a database that uses existing data files and then go to the "Modifying DB2/400 Data Definitions with Progress/400" section.

A.1.1 Creating a New Database with No Existing Data

This section explains how to create a new database with no existing data. Follow these steps:

1 ♦ Create a new, empty server schema on the AS/400 by running the DUPPRODB utility. Table A–1 notes the required DUPPRODB parameter values; additional parameters are optional.

Table A–1: DUPPRODB Parameter Values Required for No Existing Data

Parameter	Option
NEWDB(*library name*)	Enter a library name or *CURLIB.
FRMDB(*PROEMPTY)	Use this empty server schema.
OBJLIB(*library name)*	Enter the library where data files defined with the Progress/400 Data Dictionary will be located. The default is *NEWDB, which puts the data files in the same library as the server schema. To put the data files in a separate library from the server schema, enter a library name.
CRTLKT(*YES)	Enter *YES if you want Progress locking behavior. You typically assign *YES if you are not accessing existing AS/400 data files with this server schema.

See Chapter 10, "AS/400 Utilities," for more information on the DUPPRODB utility.

2 ♦ Create a client schema holder for the server schema. For more information on client utilities, see Chapter 9, "Remote Client DB2/400 Utilities."

3 ♦ Define your database using either the Progress/400 Data Dictionary on the client machine or an AS/400 tool such as DDS. See the "Modifying DB2/400 Data Definitions with Progress/400" section for more information.

A.1.2 Creating a Database Using Existing DB2/400 Data

This section explains how to create a database when your Progress/400 application uses existing data files; for example, from an RPG application. For more information, see Chapter 3, "Creating the Progress/400 Environment."

Follow these steps:

1 ♦ Create an empty server schema on the AS/400 by running the DUPPRODB utility. Table A–2 notes the required DUPPRODB parameter values; additional parameters are optional.

Table A–2: DUPPRODB Parameter Values Required for Using Existing Data

Parameter	Option
NEWDB(*library name)*	Enter a library name or *CURLIB.
FRMDB(*PROEMPTY)	Use this empty server schema.
OBJLIB(*library name)*	Enter the library where data files defined with the Progress/400 Data Dictionary will be located. The default is *NEWDB, which puts the data files in the same library as the server schema. To put the data files in a separate library from the server schema, enter a library name.
CRTLKT(*NO)	Enter *YES if you want Progress locking behavior. Enter *NO if you want OS/400 locking behavior. You typically assign *NO if you are accessing existing AS/400 data files with this server schema and those files are used with other non-Progress data files.

See Chapter 10, "AS/400 Utilities," for more information on the DUPPRODB utility.

2 ♦ Add the existing data files to the Progress/400 server schema by running the Progress/400 CHGPRODCT utility. Table A–3 notes CHGPRODCT parameter values to use.

Table A–3: CHGPRODCT Parameter Values for Using Existing Data

Parameter	Option
PRODCT(*library name)*	Enter a library name where the server schema is located or *CURLIB.
FRMFILLST	Enter a list of the data PFs that you want to access, or enter *ALL to access all files in a specific library. The utility automatically picks up the related logical files based on the Include Database Relations in this table.
From Library(s)	Enter a specific library name or enter *LIBL if the data files are in multiple libraries.
Include Database Relations	Enter *YES, the default, to preserve the logical file dependencies.
FRCCHG	Enter *YES to overwrite any Progress-specific attributes that you defined from the client ADE. Enter *NO to preserve the Progress attributes if a selected file was last updated from the Progress client. The utility does not update these files and returns an error message indicating that it did not update them.
ERRLIMIT	Enter *NOMAX to specify that the utility continue to run when an error occurs.

When using this utility, note the following:

- If you enter a specific library name for the From Library(s) parameter and want to add files from other libraries to the server schema, run CHGPRODCT again for each additional library.
- If you enter *YES to Include Database Relations, CHGPRODCT automatically picks up related logical files in other libraries without specifying the other library names.

3 ♦ Create a client schema holder for the server schema. See Chapter 9, "Remote Client DB2/400 Utilities," for more information on client utilities.

A.2 Creating a Copy of a Schema and Data

You can use the Progress/400 utilities to make a copy of your database for testing or quality assurance purposes. This process copies the following:

- Data files (empty or with data)
- The server schema
- The client schema holder

Note that the new copy of your database might include data files in one or more libraries, regardless of the original structure.

The following sections document two options for creating a copy of a schema and data:

- Copying all data files and server schema to a single OS/400 library
- Creating a copy of a database with an ALTLIB structure, where data files reside in one or more alternate libraries

For further discussion, see Chapter 2, “Common Product Information.”

A.2.1 Copying a Single Library or ALTLIB Database

This section explains how to create a database by copying all data files (physical files and logical files) and server schema to one OS/400 library. This method works whether all data files reside in the server schema library or in one or more alternate libraries. When you complete the database copy, all of the files will reside in the library that you specify.

To use this method, follow these steps:

1 ♦ Run the DUPPRODB utility to copy the server schema and data files to the new library. Table A–4 notes DUPPRODB parameter values to use.

Table A–4: DUPPRODB Parameter Values for a Library Copy

Parameter	Option
NEWDB(*new library name)*	Enter a library name or *CURLIB.
FRMDB(*old library name*)	Enter the library name containing the existing server schema.
COPYOPT(*copy selection)*	Enter *FULLCOPY to copy data files with data, or enter *EMPTYCOPY to copy data files with no data. Either option creates all copies in the server schema library, regardless of where the original copies are located.
OBJLIB(*NEWLIB)	Locate your Progress/400 Data Dictionary data files here. Select the default option, *NEWDB, which places the data files in the same library as the server schema.

2 ♦ Create a client schema holder for the server schema. See Chapter 9, "Remote Client DB2/400 Utilities," for more information on client utilities.

A.2.2 Creating a Copy of an ALTLIB Database Structure

This section explains how to create a copy of a database with an ALTLIB structure, where data files (physical files and logical files) reside in one or more alternate libraries. The resulting copy also uses multiple libraries.

To use this method, follow these steps:

1 ♦ Run the appropriate OS/400 commands to copy data files from the original library (where the data files are) to the library that will contain the copies.

2 ♦ If data exists in multiple libraries, repeat Step 1 for each library.

3 ♦ Run the DUPPRODB utility to create an empty server schema. Table A–5 notes DUPPRODB parameter values to use.

Table A–5: DUPPRODB Parameter Values for Empty Server Schema

Parameter	Option
NEWDB(*new library name)*	Enter a library name or *CURLIB.
FRMDB(*old library name*)	Enter the library name containing the existing server schema.
COPYOPT(*copy selection)*	Enter *FULLCOPY to copy data files with data, or enter *EMPTYCOPY to copy data files with no data. Either option creates all copies in the server schema library, regardless of where the original copies are located.
OBJLIB(*NEWLIB)	Locate your Progress/400 Data Dictionary data files here. Select the default option, *NEWDB, which places the data files in the same library as the server schema.

4 ♦ Run the CHGPRODCT utility for each library created in Step 1. Table A–6 notes CHGPRODCT parameter values to use.

Table A–6: CHGPRODCT Parameter Values for ALTLIB Database Structure

Parameter	Option
PRODCT(*library name)*	Enter a library name where the server schema is located or enter *CURLIB.
FRMFILLST	Enter a list of the data PFs that you want to access, or enter *ALL to access all files in a specific library. The utility automatically picks up the related logical files based on the Include Database Relations in this table.
From Library(s)	Enter a specific library name or enter *LIBL if the data files are in multiple libraries.
Include Database Relations	Enter *YES, the default, to preserve the logical file dependencies.
RTNPROATR	Enter *YES to preserve Progress data-file properties.
FRCCHG	Enter *YES to preserve non-Progress data-file properties such as select/omit.
ERRLIMIT	Enter *NOMAX to specify that the utility continue to run when an error occurs.

NOTE: If you run CHGPRODCT against data files that were created using Progress or that contain Progress-specific properties, you **must** specify the RTNPROATR parameter to retain these properties.

5 ♦ Create a client schema holder for the server schema. See Chapter 9, "Remote Client DB2/400 Utilities," for more information on client utilities.

A.3 Modifying DB2/400 Data Definitions with Progress/400

This section explains how to modify data definitions. It describes two ways to do this:

- Using an AS/400 tool such as DDS or SQL on the client.
- Using the Progress/400 Data Dictionary on the client.

Both methods update the server schema and the client schema holder. Generally, the modifications you make are interchangeable between tools. If you use an AS/400 tool to modify data definitions and then run CHGPRODCT to change data files that were originally created with Progress, you can use the RTNPROATR parameter to retain a large set of data file properties. However, if you use the Progress/400 Data Dictionary to modify data-files originally created using an AS/400 tool, non-Progress properties (for example, SELECT/OMIT criteria) are removed.

A.3.1 Modifying Data Definitions Using the Progress/400 Data Dictionary

Before you modify DB2/400 definitions that were created using the Progress/400 Data Dictionary or that contain Progress-specific properties, consider the following information:

- When you use the Progress/400 Data Dictionary in Modify Schema mode, journaling on the server schema objects (P__*) starts automatically using the journal PRODBAJRN. This allows you to roll back server schema changes before they are committed.
- **Do not** journal server schema objects to another journal. Also, you **cannot** move PRODBAJRN to another library.
- The user profile that you use to connect to the DB2/400 database must have *ALLOBJ authority to make database modifications using the Progress/400 Data Dictionary. IBM's API for Progress/400 requires this authority and cannot be changed by Progress/400.
- Do not change the object authority or location of any objects Progress/400 creates in the server schema library.

Follow these steps to modify data definitions using the Progress/400 Data Dictionary:

1 ♦ Make sure that all users disconnect from the database. The database locks until changes are either rolled back or committed.

2 ♦ Make a backup copy of your server schema and data files.

3 ♦ From the client machine, connect the client schema holder and its DB2/400 database.

4 ♦ Enter your schema changes using the Progress/400 Data Dictionary's Modify Schema mode. For more information on how to modify, add, and delete tables, fields, and indexes, see Chapter 9, "Remote Client DB2/400 Utilities."

NOTE: If you add or modify triggers that reference tables, fields, or indexes that are not committed and synchronized, you get an error message when you attempt to compile the code. This occurs when you activate the Check Syntax or Check CRC toggle box. You can save the trigger code, but you must commit transactions and synchronize the client before the code will compile successfully.

5 ♦ If you want to commit your transaction, run Edit→ Commit Transaction to update the modifications in the server schema.

NOTE: The commit process does the following on the AS/400: starts a job named APYPRODCT, creates a DDS for the new or modified table, and saves and restores data to modified tables. If anything fails during the commit process, the server schema is automatically restored to its previous state.

If you make a mistake or want to undo your changes, run Edit→ Undo Transaction.

6 ♦ Change to Read Only mode in the Progress/400 Data Dictionary.

7 ♦ From the Data Administration menu, run Synchronize Progress/400 Client to update the client schema holder.

A.3.2 Modifying Data Definitions Using an AS/400 Tool

This section explains how to modify data definitions that were created using an AS/400 tool or that contain AS/400-specific properties such as DDS keywords.

NOTE: If you use this method to modify data files, you might lose Progress data files properties when you run CHGPRODCT, unless you use the RTNPROATR parameter.

To use this method, follow these steps:

1 ♦ Run CHGPRODCT to update the server schema. Table A–7 notes CHGPRODCT parameter values to use.

Table A–7: CHGPRODCT Selections to Update Server Schema

Parameter	Option
PRODCT(*library name)*	Enter the library name where the server schema is located or enter *CURLIB.
FRMFILLST	Enter a list of the data physical files you have added, modified, or deleted. Enter *ALL to update definitions for all files in a specific library. This utility automatically picks up the related logical files if you set the Include Database Relations parameter to *YES.
From library(s)	Enter a specific library name or enter *LIBL if the data files are in multiple libraries.
Include Database Relations	Enter *YES, the default, to preserve the logical file dependencies.
PROATR	Enter *YES to preserve Progress data-file properties.
FRCCHG	Enter *SAVE to preserve non-Progress data-file properties such as select/omit.
ERRLIMIT	Enter *NOMAX to specify that the utility continue to run when an error occurs.

When using this utility, note the following:

- If you enter a specific library name for the From Library(s) parameter and want to update definitions for data files in other libraries, run CHGPRODCT again for each additional library.
- If you enter *YES to Include Database Relations, CHGPRODCT automatically picks up related logical files in other libraries without specifying the other library names.

2 ♦ Create a new client schema holder or synchronize an existing one for the server schema. See Chapter 9, "Remote Client DB2/400 Utilities," for more information on client utilities.

B

Legacy Support and the Progress Unknown Value

Progress/400 provides backward compatibility to support prior DB2/400 translations of the legacy unknown value. With Version 9 of the DataServer, you can convert data definitions that used the legacy unknown value to use the SQL NULL value or you can continue to use the legacy unknown value. This appendix describes the following topics related to SQL NULL and the legacy unknown value support:

- Legacy unknown value support
- Converting your data definitions from legacy unknown values to SQL NULL values
- Using the Windows client to manage SQL NULL value assignments
- How Progress/400 handles SQL NULL and legacy unknown values in a query

Progress recommends the use of DB2/400 SQL NULL capability

B.1 Legacy Unknown Value

When the user does not use SQL NULL support, the Progress client translates the question mark (?) to the legacy unknown value as follows:

- When the user assigns the question mark (?) to a field, the Progress client assigns a value to the DB2/400 field based on the data type of the field. Table B–1 describes the implementation of the unknown value for each data type.
- When a Progress application retrieves a field that contains an unknown value, it returns the standard Progress unknown value (?).

Table B–1 lists the possible unknown values by data type.

Table B–1: The Legacy Unknown Value Implementation by Data Type

Data Type	Implementation
CHARACTER	A question mark (?) followed by as many blanks as necessary to fill out the field. For a field length of 1, the value is binary 255.
DATE	The AS/400 date data type. See Table B–2 for the unknown values for the various date formats.
INTEGER	2-byte: 32,767 (the largest possible value) 4-byte: 2,147,483,647
LOGICAL	A question mark (?).
DECIMAL	An invalid packed decimal value with an invalid sign.[1]

[1] The unknown packed decimal value causes errors when other languages, such as RPG, access it.

Unlike a standard Progress database, DB2/400 allows only one record with the legacy unknown value in the index field when the index is defined as a unique index.

When you load Progress DATE data that contains the unknown value (?) into OS/400 physical files, the DataServer inserts values based on the DB2/400 date format for the field. Table B–2 lists the values that the DataServer inserts.

Table B–2: Legacy Unknown Value Defaults for DB2/400 Date Formats

DB2/400 Storage Format	Unknown Value
*EUR	31.12.9999
*ISO	9999-12-31
*JIS	9999-12-31
*USA	12/31/9999
*DMY	31/12/99
*MDY	12/31/99
*YMD	99/12/31
*JUL	99/365

B.2 DUPPRODB Legacy Unknown Value Support

The DUPPRODB utility will adopt existing support of the unknown value. See Chapter 10, "AS/400 Utilities," for a detailed description.

B.3 Converting from Legacy Unknown Values to SQL NULL Support

Follow these steps to convert your data and definitions from legacy unknown value support to SQL NULL support:

1 ♦ On your Windows client, dump both your data definitions (.df) and data (.d).

2 ♦ On the AS/400, run DUPPRODB FRMDB (*EMPTY).

3 ♦ On your Windows client, reload the data definitions (.df) with the Progress/400 Data Dictionary, accepting the default to allow fields to be null capable.

NOTE: This operation creates all data files and fields as SQL NULL enabled.

4 ♦ On your Windows client, reload the data (.d) once the data definitions are loaded.

NOTE: Fields that contain the unknown value are stored as SQL NULL fields in the data files.

B.4 Using the Windows Client to Manage SQL NULL Value Assignments

If you use the Progress/400 Data Dictionary to maintain your DB2/400 files, two options that affect the outcome of legacy unknown value assignments or SQL NULL assignments are available to you.

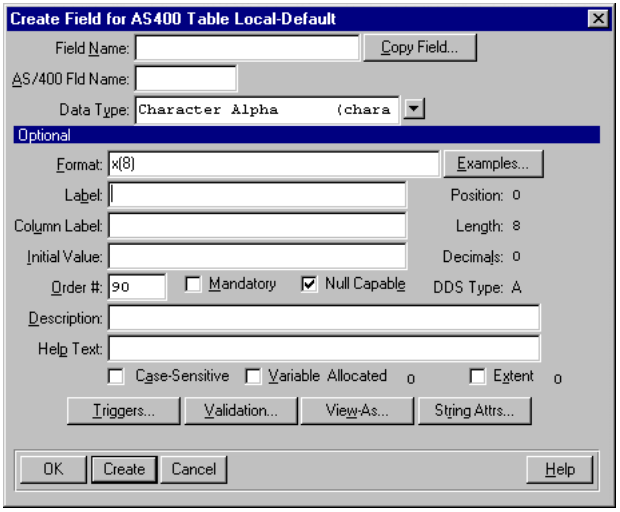
Follow these steps to set these two values with the Progress/400 Data Dictionary:

1 ♦ Choose the Field mode button in the Progress/400 Data Dictionary main window.

2 ♦ Choose Modify schema.

3 ♦ Select the table that will contain the new field from the Tables list.

4 ♦ Choose the Create Field button. The following dialog box appears:



You can select the Mandatory or the Null Capable option setting. Table B–3 displays possible settings for these two values. Table B–3 also describes which assignments are allowed based on these option settings.

Table B–3: Progress/400 Data Dictionary Options for SQL NULL and Unknown Value Assignments

Mandatory	NULL Capable	Result
ON	ON	4GL prevents (?) assignment for unknown value.
OFF	ON	Any value is legal.
ON	OFF	4GL prevents (?) assignment.
OFF	OFF	Any value is legal.

B.5 How Progress/400 Handles SQL NULL and Legacy Unknown Value Support

Progress queries and SQL queries on the AS/400 can return different values. As a comparison, Table B–4 shows the return values for queries based on your DUPPRODB and NULL capable selections.

Table B–4: SQL and 4GL Query Results with NULL Capable and ALWPROUNK Settings

Progress/400 Data Dictionary NULL Capable	SQL Query Result	4GL Query Result
YES	NULL	Returns unknown value (?)
NO	Legacy unknown value	Returns unknown value (?)
NO	N/A	OS/400 data mapping error

C

Useful OS/400 Commands

This appendix documents a number of OS/400 CL commands that you might find useful when working with the Progress/400 DataServer.

C.1 OS/400 CL Command List

Table C–1 lists and describes useful OS/400 CL commands.

Table C–1: Useful OS/400 Commands

Command	Usage
CLRPFM	Clears a physical file member of its contents.
CPYF	Copies a file structure to another file. Duplicate data is an option.
CRTDUPOBJ	Creates a duplicate object. Duplicate data is an option.
CRTLIB	Creates an OS/400 library.
CRTLF	Creates a logical file.
CRTPF	Creates a physical file.
DLTLIB	Deletes an OS/400 library.
DSPJOBLOG	Displays a job's program message logging file.
DSPPFM	Displays the contents of a physical file member.
EDT/GRT/RVK:OBJAUT	Edits, grants, and revokes object authority.
WRK:F/OBJ/LIB	Works with files, objects, and libraries.
WRKACTJOB	Works with all currently active jobs on the system.
WRKOUTQ	Works with out-queue (spooled files for completed jobs).
WRKSPLF	Works with spooled print files associated with the user.

For more information on any of these parameters, see your IBM AS/400 documentation.

Glossary

Absolute Path

Used with the IFS file system, absolute paths include the complete path to an object. Absolute paths always begin with a slash (/).

AdminServer

A process that manages the AppServer and DataServer products.

Advanced Program-to-Program Communications (APPC)

The communications support that allows AS/400 applications to communicate with other programs over a network.

Application Service

An arbitrary designation for the business function that the Unified Broker instance provides.

AppServer

A client of the Unified Broker product that includes 4GL clients and Open Clients.

Array

A field or variable with multiple elements. Each element in an array has the same data type. The Progress/400 DataServer supports arrays by using a set of similar fields that all have the same type and same length.

Authority List

An AS/400 object that lists the users or groups that have access to objects, and the type of access those users have.

Auto-connect

A Progress feature that uses information stored in a schema holder to connect to the database it describes when that database is referenced in a compiled application at run time.

Buffer

A piece of memory used as a temporary storage area for data during input or output operations. See also **Data Buffer**.

Code Page

Contains the numeric codes assigned to the various characters when they are stored in databases.

Collation

Specifies the order in which characters are sorted for indexes, queries, and so forth; that is, the collation sequence. A collation table contains this sort-order information.

Commitment Control

A method of grouping file operations so that a group of changes is either implemented (through the COMMIT command) or removed (through the ROLLBACK command) as a single unit. Commitment control provides journal entries with transaction boundaries.

Connection Parameters

Variables or constants used to control how Progress connects to a database.

Data Buffer

Progress uses two types of data buffers: the record buffer, which is a temporary storage area for a record, field, or variable; and the screen buffer, which is a display area for a field, variable, or the result of a calculation.

Data Definitions

The characteristics of the files, fields, and indexes that comprise the schema of a Progress database. The structure of a given database. See also **Data Dictionary**.

Data Description Specifications (DDS)

An AS/400 tool used to define files.

Data Dictionary

The Progress/400 Data Dictionary tool that allows you to maintain data definitions on the AS/400.

Data Type

A property of a field or variable that determines the nature of data that can be stored and manipulated (integers or characters, for example).

Database Administrator (DBA)

The individual responsible for maintaining a database management system.

Database Administrator (DBA) Privileges

The level of database access required for database administrators.

Date Data Type

In Progress, a date data type can contain a date and time from 1/1/32768 B.C. to 12/31/32767 A.D.

Date Field

A field that has a date data type. A date field can contain dates from 1/1/32768 B.C. through 12/31/32767 A.D. You can specify dates in this century with either a two-digit year, such as 8/9/90, or a four-digit year (8/9/1990). Dates in other centuries require a four-digit year.

Decimal Data Type

A property of a field or variable that determines the data that can be stored there can be a decimal.

Decimal Field

A field that has a decimal data type. A decimal field can contain up to 50 digits.

Dictionary Library

An AS/400 Dictionary Library is a logical object built by Progress /400. It contains the Progress/400 Server Schema and Schema Image. It can optionally contain application data.

Distributed Data Management (DDM)

DDM files are like pointer objects. They point to actual physical or logical files on a remote AS/400.

Environment Variable

System variables used to tailor a user's working environment; for example, to set search paths for files and set up terminal definitions.

Error Log

A file that contains the errors that Progress finds during program execution.

Extent

1) The number of elements in an array field or variable.
2) One of the volumes in a multi-volume database.

Field

A component of a record that holds a data value.

File

A collection of logically related records treated as a unit. A file can contain a program, data, or text. For example, a payroll file is a collection of employee payroll records.

Host Language Interface (HLI)

A Progress product for embedding SQL statements into host language applications and retrieving or updating records from Progress and non-Progress databases.

Index

A directory or table that contains the fields identifying the records in a file and the locations where the records are stored.

Integer Data Type

A property of a field or variable that allows the storage of integers. In Progress, an integer can be a positive or negative whole number, within the range of -2,147,483,648 through 2,147,483,647.

Integer Field

A field that has an integer data type. In Progress, an integer field can contain positive or negative whole numbers, ranging in value from -2,147,483,648 through 2,147,483,647.

Integrated File System (IFS)

An umbrella file system on the AS/400 that can access all other file systems. You must use IFS to access the ROOT file system.

Job

User work on the AS/400. It can be interactive or batch. Each job has its own name and characteristics.

Job Attributes

Define how jobs execute on the AS/400.

Journal

An AS/400 object (*JRN) made up of journal entries.

Journal Entries

Records the DB2/400 database writes when you change database records.

Journaling

Recording changes made to files. Used to reconstruct a file by applying the journaled changes to a saved version of the file.

Journal Receiver

An AS/400 object (*JRNRCV) that holds a journal.

Library

An AS/400 object (*LIB). It is the AS/400 unit of organization.

Library List

An AS/400 object (*LIBL). It is used as a search path for objects on the system.

Locks

A restriction on data access so that updates can be performed without compromising data integrity.

Logical Data Type

One of two values consisting of yes/no, true/false, or logical value pairs.

Logical Database Name

The name used to refer to a database in Progress code. The logical database name is stored in a procedure's object code. Usually, this identifies a database with an identical name, but another name can be specified when connecting to the database. When a procedure executes, its database name references must match the logical names of connected databases.

Logical Field

A field that has a logical data type. A logical field can contain one of two values.

Logical File

An AS/400 object type (*FILE(sub-type LF)) that contains an alternate view of data. The Progress/400 DataServer uses logical files to implement secondary indexes.

Logical Unit (LU)

A logical device for accessing the communications network.

Name Server

A Name Server is a single process that mediates client connections for a set of Unified Brokers that have registered with it.

Native 4GL Client

A product local to the AS/400 that runs Progress applications. The Native 4GL Client does not have interactive capabilities. All output must be redirected to a file or printer.

Network Address

The way each physical and logical unit is identified to the network and other objects on the network.

N-tier

A computing model that supports a flexible networking structure. You can physically distribute application logic and processing loads among many machines across a distributed network. Also, the n-tier model supports the logical separation of user interface, application logic, and data across three or more machines.

Object

The AS/400 operating system (OS/400) identifies anything that exists and occupies spaces in storage as an object.

Object Authority

The permission to access an object.

Parameter

A variable or constant passed to or from a subroutine and the main program. Also referred to as a Progress startup parameter.

Parameter File

A file containing Progress startup parameters. Parameter files store the appropriate startup (connect) parameters for a particular database, group of users, or system configuration, rather than supplying startup or CONNECT options explicitly on the Progress command line or in a CONNECT statement.

Partner Logical Unit (PLU)

The secondary LU you communicate with over the SNA network during an LU-to-LU communication session. The primary LU is your own LU.

Physical Database Name

The actual name of a database.

Physical File

An AS/400 object type (*FILE(sub-type)) that contains data or source code.

Physical Unit (PU)

The physical devices that connect network users.

Place Holder File

A Progress/400 database file object that serves as a representative for a DB2/400 file object.

QCMD

A data definition file that contains a parameter form for the OS/400 Submit Job (SBMJOB) command.

Record

An ordered set of fields that make up a file.

Remote Machine

A machine that is connected through a network and is not your local machine.

ROOT File System

A file system on the AS/400 that mimics UNIX (/) and Windows (\) style paths.

Schema

A definition of the structure of a database (for example, the files it contains, the fields within the files, views). In addition to database structure, Progress database schemas contain items such as validation expressions and validation messages.

Schema Holder

A Progress database that contains the schema definitions for one or more non-Progress databases, typically located on the client machine.

Schema Image

A P__SCHEMA file that is added to the server schema on the AS/400. This file contains schema definitions for the Native 4GL Compiler.

Startup Parameters

Parameters that tailor a Progress session or database connection.

Stored Procedure

Any program object created with any of the available languages on OS/400: CL, RPG/400, ILE RPG/400, COBOL/400, ILE COBOL/400, ILE C/400, PL1/400, FORTRAN/400, or REXX.

Stream File

A file that does not get broken down into individual records with fixed sizes. Stream files can be variable length and have different data formats. EBCDIC stream files might be text files, such as p-code. Binary stream files might be pictures or object code, such as r-code.

Structured Query Language (SQL)

A database language that consists of a set of facilities for defining, manipulating, and controlling data in a relational database.

Subsystem

An AS/400 operating environment that controls a set of jobs currently being processed.

Subsystem Attributes

Define the characteristics of the subsystem.

Subsystem Description

An AS/400 object (*SBSD) where you define subsystem attributes.

Systems Network Architecture (SNA)

The structure used in networks that defines the formats and protocols used to transmit information through networks.

Target Database

The Progress or non-Progress database or files that you want to access data from when you run a procedure.

Transaction

A set of changes to the database, which should either be completely done or should leave no modification to the database.

Transaction Scoping Rules

The set of rules that determines the range of a transaction.

Unified Broker

Manages connections between clients and the Unified Broker instance, registers broker information with the controlling NameServer, and maintains the status of each 4GL process running an AppServer.

Unified Broker Properties File

Progress stores the configurations for all the component configuration management by the Unified Broker administration framework in a properties file named `ubroker.properties`.

Unknown Value

A special Progress data value (represented by a question mark (`?`)) that indicates that the field or variable data is unknown or unavailable.

User ID

User identification.

User Profile

Identifies you to the AS/400 system. It includes such information as what objects you can access and how the system handles jobs you submit.

Validation Expression

A test to validate data before storing it in a field.

Index

A

Absolute paths 5–3

Accessing DB2/400 database files 1–9
  overview 3–2

Adding items to Progress/400 Data Dictionary
  fields 9–10
  indexes 9–14
  sequences 9–17
  tables 9–7

Admin menu of Progress/400 Data Dictionary utility 9–24

After-image file 8–6

Aliases 5–6

ALLOW-SELECT-OMIT-INDEXES setting 8–24

ALLOW-ZONE-DECIMAL setting 8–24
  usage notes 8–27

ALTER TABLE statement 2–25

ALTSEQ tables 8–12, 8–18
  corresponding code pages 8–16
  creating 8–19
  DataServer considerations 8–21

ALTSEQCI table 8–18

APIs
  chgdara.p 11–22
  chguspc.p 11–28
  rcvdtaqe.p 11–36
  rtvdara.p 11–24
  rtvuspc.p 11–30
  snddtaqe.p 11–33

AppBuilder 1–1

Applications
  running on the AS/400 1–10
  security 1–11, 2–27

AppServer 1–1
  architecture 1–3
  security 8–4

AppServer client, directing output to printers 5–9

Architectures
AppServer 1–3
client/server 1–2, 1–4
DDM 2–7
Native 4GL Client 1–3
n-tier 1–2
Web-based 1–2

AS/400
migrating Progress databases 1–9
operating-system code page 8–10
running Progress applications 1–10
stream-file code page 8–11
transferring p-code 5–10

ASCII, collation sequence 2–5

Attributes, retaining Progress 8–28

Audience xvii

Auto-connect 4–16, 4–17

B

Before-image file 4–9

Before-imaging, local 8–4

Binary data type 2–10

BOF processing. *See* Beginning-of-file (BOF) processing

Bold typeface, as typographical convention xx

Brokers
managing TCP/IP 4–5
TCP/IP 4–3

C

Canceling query requests
connection parameter 4–10

CANCELQUERY argument to -Dsrv parameter 4–10

Case insensitivity 2–4
collation table 8–18

CCISD parameter of SBMJOB command 11–15

CHANGE DATA AREA API. *See* chgdara.p API

CHANGE USER SPACE API. *See* chguspc.p API

CHANGE-CHAR-TO-VARLEN setting 8–26

Changing
connection information 9–4
data
in OS/4 data areas 11–22
in OS/4 user spaces 11–28
data definitions 3–4

CHARACTER data type
and unknown value B–2

Character data type 2–10, 2–11

Checking logical file level
connection parameter 4–10

chgdara.p API 11–22
coding example 11–26
parameters 11–22
return values 11–23

CHGJRN command 9–6

CHGPRODCT utility 3–4, 10–8
parameters 10–9
PROATR keyword/attribute interaction 8–29

CHGPRODFT utility 10–11

chguspc.p API 11–28
coding example 11–32
parameters 11–28
return values 11–29

Client schema holder. *See* Schema holder

Client utilities 9–1
See also Utilities

Client/server architecture 1–2

Clients
AppServer 5–1, 5–6
compiling code 5–4
-cp* startup parameters 5–12
creating schema image 5–4
directing output to printers 5–9
executing code 5–4, 5–6
executing OS/400 commands 11–8
IFS 5–2
input and output 5–6
internationalizing 5–12
native 5–1
Native 4GL 1–3
Progress, executing OS/400 commands 11–8
QCMD 5–10
remote connection 4–3
task list for creating and using 5–4
transferring 4GL code 5–4

CLOSE command 11–9

CLOSTMF command 5–10

CMD parameter of SBMJOB command 11–13

Code pages 8–9
AS/400 operating system 8–10
AS/400 stream files 8–11
corresponding ALTSEQ tables 8–16
DataServer 8–10
identifying 8–10
Native 4GL Client 8–11
overview 8–9
Progress client 8–11
Progress/400 databases 8–11
schema holder 8–12
servers 8–10
supported in Progress/400 8–8
Windows operating system 8–11
Windows stream files 8–12

Collation 8–9
See also ALTSEQ tables
ALTSEQCI table 8–18
case-insensitive table 8–18
LOCASE table 8–19
lowercase table 8–19
sequences 2–5, 8–9
setting up for Progress databases 8–13
supported in Progress/400 8–8
table example 8–19
table sort order 8–20
tables 8–9, 8–20, 10–1
UPCASE table 8–18
uppercase table 8–18
user-defined 8–17

Commands
ADDPFTRG 2–33
APYJRNCHG 2–33
CLOSE 11–9
CLOSTMF 5–10
CRTJRN 8–7
CRTJRNRCV 8–7
CRTTBL 8–19
DLTJRN 8–7
DLTJRNRCV 8–7
DSPJOB 11–8
DSPJRN 8–7
ENDJRNPF 8–7
EPI 11–2
EPI CALL 11–3
EPI ENTRY 6–6
EPI EXIT 6–6
journaling 8–7
OPNSTMF 5–10
OS/400, executing from Progress client 11–8
recovering corrupted word indexes 2–33
RGZPFM 2–33
RMVJRNCHG 2–33
RMVPFTRG 2–33
RSTLIB 2–34
RSTOBJ 2–34
SBMJOB 11–8
STRJRNPF 8–7
WRKJRNA 8–7
WRTSTMF 5–10

Commitment control 8–4
See also Transaction control
connection parameter 4–9
locking records 2–16

COMMITMENT-CONTROL-SCOPE
setting 8–23

Communications protocols
SNA 4–5
TCP/IP 4–3

Compiling p-code 5–11

COMPRESS argument to -Dsrv parameter 4–9

Compression, connection parameter 4–9

CONNECT statement 4–16, 4–17

Connecting to a DB2/400 database 4–15
at startup 4–16
connection modes 4–18
from the command line 4–16
general considerations 4–17
logical database names 4–17
SETUSERID function 2–27
troubleshooting 4–19
USERID function 2–27
using SNA and auto-connect 4–17
using SNA and CONNECT 4–17
using TCP/IP and auto-connect 4–17
using TCP/IP and CONNECT 4–16

Connection failures
error messages 4–20
Progress responses 4–19

Connection modes
multi-user 4–18
single-user 4–18

Connection parameters 4–6
changing 9–4
Database Type (-dt) 4–12
DataServer (-Dsrv) 4–8, 8–22
Host Name (-H) 4–12
Ignore Stamp (-is) 4–13
Network Type (-N) 4–13
Password (-P) 4–14
Physical Database Name (-db) 4–8
Profile Name (-H) 4–12
security 8–3
Server Name (-Sn) 4–15
Service Name (-S) 4–14
User ID (-U) 4–14

Constructing the PROPATH value 5–11

CONVMAP.DAT file 8–21

Corrupted word indexes, recovering 2–33

-cp* startup parameters 5–12

Create DB2/400 DataServer Schema utility 9–2

Create Incremental .df Files utility 9–28

CREATE INDEX statement 2–25

CREATE TABLE statement 2–25

CREATE VIEW statement 2–25

Creating
ALTSEQ tables 8–19
incremental .df files 9–28
journal receiver 8–7
lock table 2–19
production environments 8–31
Progress/400 environment 3–3, 3–11, 5–5
records
duplicate key 2–17
RECID function 2–26
ROWID function 2–26
schema holder 3–7, 3–15, 9–2
schema image 5–4
server schema 3–3, 3–11
test environments 8–31

CRTJRN command 8–7

CRTJRNRCV command 8–7

CRTPROLKT utility 2–19, 10–13
  parameters 2–19, 10–13

CRTSCHIMG utility 5–5, 10–13
  parameters 10–13

CRTTBL command 8–19

CURLIB parameter of SBMJOB command 11–14

CVTPRODCT utility 10–14
  parameters 10–14

CVTSRVSCH utility 10–15
  parameters 10–15

D

Data
  accessing
    in OS/400 data areas 11–21
    in OS/400 data queues 11–32
    in OS/400 user spaces 11–27
  changing
    in OS/400 data areas 11–22
    in OS/400 user spaces 11–28
  definitions
    dumping 9–25
    loading 9–30
  dumping 3–12, 9–26
  loading 3–17, 9–32
  receiving in OS/400 data queues 11–36
  retrieving
    from OS/400 data areas 11–24
    from OS/400 user spaces 11–30
  sending to OS/400 data queues 11–33

Data Administration tool 9–2

Data areas, OS/400 11–21
  changing data 11–22
  coding example 11–26
  QDTAARA table 11–21
  retrieving data 11–24

Data compression, connection parameter 4–9

Data definitions
  changing 3–4
  dumping 3–12
  loading 3–17
  maintaining 3–9, 3–18

Data Dictionary 1–1
  See also Progress/400 Data Dictionary

Data library, connection parameter 4–9

Data queues, OS/400 11–32
  coding examples 11–38
  QDTAQ-ENTRY table 11–33
  receiving data 11–36
  sending data 11–33
  using LOOKUP and SUBSTRING functions 11–35

Data types 2–10
  binary 2–10
  character 2–10, 2–11
  DATE 2–13
  date formats 2–13
  DECIMAL 2–13
  decimal 2–10
  floating point 2–11
  INTEGER 2–14
  LOGICAL 2–14
  mapping
    DB2/400 to Progress 2–10
    Progress to DB2/400 2–11
    Progress to OS/400 11–5
  packed decimal 2–10
  selecting in Data Dictionary 9–11
  time 2–11
  timestamp 2–11
  zoned decimal 2–10

Database
accessing an existing 3–3
DUPPRODB 8–14
moving to the AS/400 3–10

Database objects
DB2/400 2–2
invalid characters in names 2–3
naming restrictions 2–3
Progress 2–2
unique names 2–4

Database Type (-dt) connection parameter 4–12

Databases
accessing DB2/400 files 1–9, 3–2
connection modes 4–18
DB2/400, connecting to 4–15
DB2/400, disconnecting from 4–22
demonstration 1–8
design issues 2–2
logical names 4–17
migrating to the AS/400 1–9
Progress/400, code page 8–11
recovering 8–6
security 8–2
setting up collation 8–13
transaction control 8–4
triggers 2–5
using SETUSERID when connecting 2–27
using USERID when connecting 2–27

DATALIB argument to -Dsrv parameter 4–9

DataServer
internal code page 8–10
logic 4–2
performance tuning 8–21
security 1–11
upgrading to current version 1–10
usage guidelines 1–9
using ALTSEQ tables 8–21
utilities 1–8

DataServer (-Dsrv) connection parameter 4–8
arguments 4–9

DATE data type 2–13
and unknown value B–2

Date formats 2–13

DATE parameter of SBMJOB command 11–15

-db connection parameter 4–8

DB2/400
accessing database files 1–9, 3–2
connecting to 4–15
data types, mapping to Progress 2–10
database objects 2–2
definition xvii
disconnecting from 4–22
dumping
data 9–26
data definitions 9–25
sequence current values 9–28
sequence definitions 9–27
invalid characters in object names 2–3
loading
data 9–32
data definitions 9–30
sequence current values 9–33
locks 2–20
maintaining data definitions 3–9, 3–18
naming conventions 2–3

DBA mode 9–6

DBNAME function 2–25

DDS
maintaining data definitions 3–18

DDS, maintaining data definitions 3–10

DECIMAL data type 2–13
and unknown value B–2

Defining
field length 9–12
user ID and password at startup 2–28

Definition files
loading 8–22

Delete DB2/400 DataServer Schema utility 9–5

Deleting
fields 9–14
indexes 9–17
lock table 2–19
schema holder 9–5
sequences 9–18
tables 9–9

Demonstration databases 1–8

Deploying a schema holder 3–9

Dictionary library 1–5

Dictionary objects 1–7

Disconnecting from a DB2/400 database 4–22

Distributed Data Management 2–7
architecture 2–7
files 2–7
level checking 2–10
schema holder 2–10

Distributed Database Management files. *See* DDM files

DLTJRN command 8–7

DLTJRNRCV command 8–7

DMPRODTA utility 10–21, 10–22
parameters 10–22

Documentation
IBM 1–13
Progress 1–13
usage guidelines 1–11

DROP INDEX statement 2–25

DROP TABLE statement 2–25

DROP VIEW statement 2–25

DSPJOB command 11–8

DSPJRN command 8–7

DSPSMBJOB parameter of SBMJOB command 11–15

-Dsrv connection parameter 4–8, 4–9, 8–22

-dt connection parameter 4–12

Dump Data & Definitions utility 9–25

Dump Sequence Definitions utility 9–27

Dump Sequences Current Values utility 9–28

Dump Table Contents utility 9–26

Dumping
data 3–12, 9–26
data definitions 3–12, 9–25
sequence current values 9–28
sequence definitions 9–27

Duplicate keys 2–17

DUPPRODB utility 3–3, 3–11, 5–5, 8–14, 10–15
creating test and production environments 8–31
duplicating Progress/400 Data Dictionary 8–31
parameters 10–16

E

EBCDIC, collation sequence 2–5

Edit DB2/400 Connection Information utility 9–4

ENDJRNPF command 8–7, 8–8

ENDPRONET utility 10–31
parameters 10–31
stopping TCP/IP brokers 4–5

ENDWISPRC utility 10–26

EOF processing. *See* End-of-file (EOF) processing

EPI CALL command 11–3
  parameters 11–4
  syntax 11–4

EPI command 11–2
  error codes 11–2
  parameters 11–2
  syntax 11–2

EPI ENTRY command 6–6

EPI EXIT command 6–6

Error codes, EPI command 11–2

Error handling, QCMD interface 11–12

Error messages, displaying descriptions xxiv

EXCLUDE-NULL-KEY-VALUE setting 8–26

EXCLUSIVE-LOCK state 2–18
  access conflicts 2–20, 2–22

Executing, SBMJOB command 11–10

Extent fields, and word indexes 2–35

External Programming Interface. *See* EPI command

F

Field lists 2–23

Field values, unknown 2–14

Fields
  adding 9–10
  defining length 9–12
  deleting 9–14
  format length 9–12
  modifying 9–13
  properties 9–13

File I/O between QYSLS.LIB and ROOT file systems 5–6

File keys. *See* Keys

File location, connection parameter 4–9

Files
  See also DDM files, Logical files, Physical files
  CONVMAP.DAT 8–21

File-transfer commands 5–10

FIND statement 8–22

Floating point data type 2–11

FOR EACH statement, record prefetch 2–23

Freezing tables 9–35

G

GENWRDIDX utility 10–26
  parameters 10–27

GET NEXT statement 8–22

GRANT statement 2–25

H

-H connection parameter 4–12

Help, Progress messages xxiv

HOLD parameter of SBMJOB command 11–15

Host Name (-H) connection parameter 4–12

I

IFS 5–2
ROOT file system 5–2

Ignore Stamp (-is) connection parameter 4–13

INCLUDE-VIRTUAL-FILES setting 8–27

Incremental .df files 9–28

Indexes 2–4
adding 9–14
case-insensitive 2–4
collation sequences 2–5
deleting 9–17
modifying 9–16
primary 2–4
unknown value B–2
word 2–28

INLLIB parameter of SBMJOB command 11–14

INQMSGRPY parameter of SBMJOB command 11–15

INTEGER data type 2–14
and unknown value B–2

Integrated File System
See also IFS
OUTPUT To statement 5–8

Internationalizing the client 5–12

-is connection parameter 4–13

Italic typeface, as typographical convention xx

J

Job control 8–2

JOB parameter of SBMJOB command 11–13

JOBD parameter of SBMJOB command 11–13

JOBPTY parameter of SBMJOB command 11–14

JOBQ parameter of SBMJOB command 11–13

Joined logical files 2–5
queries against 2–22

Journal entries 8–6

Journal receiver 8–4, 8–6
creating 8–7
maintaining 8–8

Journaling 8–4, 8–6
CL commands 8–7
creating a journal receiver 8–7
maintaining a journal receiver 8–8

JRNRCV object. *See* Journal receiver

K

Keys 2–4
duplicate 2–17

Keystrokes xx

L

Language restrictions, Progress 4GL 2–25

LBI keyword 8–5

LFLVLCHK argument to -Dsrv parameter 4–10

Limited logical files 2–5

Load Data & Definitions utility 9–30

Load Sequences Current Values utility 9–33

Load Table Contents utility 9–32

Loading
data 3–17, 9–32
data definitions 3–17, 9–30
definition files 8–22
load utility error (.e) files 9–33
sequence current values 9–33

Local before-image file 8–4

LOCASE table 8–19

Lock table 2–18
creating 2–19
deleting 2–19
maintaining 2–19
modifying 2–19
re-creating after AS/400 crash 2–22
updating 2–19

Locking
access conflicts 2–20, 2–22
DB2/400 locks 2–20
EXCLUSIVE-LOCK 2–18
multiple defined buffers 2–21
NO-LOCK 2–18
OS/400 object-level lock states 2–21
programming considerations 2–20
Progress and non-Progress applications 2–20
Progress locks 2–18
records 2–18
SHARE-LOCK 2–18
wait time for records 2–21

LOGCLPGM parameter of SBMJOB command 11–14

LOGICAL data type 2–14
and unknown value B–2

Logical database names 4–17
changing 9–4

Logical file level checking method connection parameter 4–10

Logical files 2–5
built over multiple physical file members 2–6
joined 2–5
limited 2–5
multiple record 2–5

Lowercase collation table 8–19

M

Maintaining
DB2/400 data definitions 3–9, 3–18
journal receiver 8–8
lock table 2–19

Managing TCP/IP brokers 4–5

Manual
organization of xviii
syntax notation xxi

MAP-VARIANT-CHARACTERS setting 8–24

Message Buffer Size (-Mm) startup parameter 4–18

Messages, displaying descriptions xxiv

Metaschema 1–5

Migrating Progress databases to the AS/400 1–9
task list 3–10

-Mm startup parameter 4–18, 8–22

MNGPRONET utility 10–32
managing TCP/IP brokers 4–5
parameters 10–32

Modifying
fields 9–13
indexes 9–16
lock table 2–19
Progress/400 environment 5–5
sequences 9–18
tables 9–8

Monitoring TCP/IP brokers 4–5

Monospaced typeface, as typographical convention xx

Moving Progress databases. *See* Migrating Progress databases

MSGQ parameter of SBMJOB command 11–15

Multiple members
  accessing data 11–12

Multiple record logical files 2–5

Multi-user connection mode 4–18

N

-N connection parameter 4–13

Naming conventions 2–3
  invalid characters 2–3

Native 4GL Client 1–1, 1–3
  Architecture 1–3
  compiling p-code 5–11
  constructing PROPATH value 5–11
  directing output to printer 5–9
  internal code page 8–11
  using aliases 5–6
  utilities 10–34

Network traffic, performance tuning 8–22

Network Type (-N) connection parameter 4–13

Networking utilities 10–31

NO-LOCK state 2–18
  record prefetch 2–23

N-tier architecture 1–2

Null value B–2

O

Object authorities
  CHGJRN command 9–6
  required 8–3

Objects 1–5

Online help for server utilities 10–5

OPNSTMF command 5–10

OS/400
  calling programs from Native 4GL Client 11–3
  EPI CALL command 11–3
  EPI command 11–2
  IFS (Integrated File System) 5–2
  interface with Native 4GL Client 11–2
  job numbers 11–11
  journaling 8–4, 8–6
  security 1–11

OS/400 objects
  accessing from Progress 4GL 11–20
  data areas 11–21
  data queues 11–32
  user spaces 11–27

OUTPTY parameter of SBMJOB command 11–14

OUTQ parameter of SBMJOB command 11–14

OVRDBF command 11–11

P

-P connection parameter 4–14

P__files, descriptions 1–6

Packed decimal data type 2–10

Parameters
See also Connection parameters, Startup parameters
chgdara.p API 11–22
chguspc.p API 11–28
rcvdtaqe.p API 11–36
rtvdara.p API 11–24
rtvuspc.p API 11–30
SBMJOB command 11–13
snddtaqe.p API 11–33

Password (-P) connection parameter 4–14, 8–3

Password, defining at startup 2–28

P-code 5–10, 5–11

Performance tuning 8–21
network traffic 8–22
queries 8–21

Physical Database Name (-db) connection parameter 4–8

Physical files 2–5
building logical files over multiple members 2–6

Prefetch 8–22

Primary index 2–4

PROCALL program 6–4
parameters 6–5

PROCESS-DFT-DDS-KEYWORD setting 8–25

PRODEMO demonstration database 1–8

Production environments, creating 8–31

Profile Name (-H) connection parameter 4–12

Program authorities, required 8–3

Progress
data types, mapping
to DB2/400 2–11
to OS/400 11–5
database objects 2–2
locks 2–18
migrating databases to the AS/400 1–9
naming conventions 2–3
retaining attributes 8–28
running applications on the AS/400 1–10
setting up collation for databases 8–13

Progress 4GL
ALTER TABLE statement 2–25
CREATE INDEX statement 2–25
CREATE TABLE statement 2–25
CREATE VIEW statement 2–25
DBNAME function 2–25
DROP INDEX statement 2–25
DROP TABLE statement 2–25
DROP VIEW statement 2–25
executing code from clients 5–6
GRANT statement 2–25
PROCALL program 6–4
RECID function 2–26
restrictions on use 2–25
REVOKE statement 2–27
ROWID function 2–26
SETUSERID function 2–27
USERID function 2–27
word-index coding issues 2–34

Progress client
code pages 8–11
internal code page 8–11
local before-imaging 8–4
synchronizing 9–33

Progress database
using as schema holder 3–13

Progress database, using as schema holder 3–5

Progress/400
code page and collation support 8–8
connection parameters 4–7
DDM 2–7
dictionary objects 1–7
internals 1–4
jobs, managing 8–2
objects 1–5
product set 1–9
PROSET settings file 8–22
reserved words 8–30
Synchronize utility 9–33
system administration 8–1
unknown value support B–2, B–3
using 1–9
utilities 1–8
word indexes 2–28

Progress/400 Data Dictionary
adding
fields 9–10
indexes 9–14
sequences 9–17
tables 9–7
Admin menu utilities 9–24
DBA requirements 9–6
defining field length 9–12
deleting
fields 9–14
indexes 9–17
sequences 9–18
tables 9–9
duplicating 8–31
field format length 9–12
maintaining data definitions 3–9, 3–18
modes 9–6
modifying
fields 9–13
indexes 9–16
sequences 9–18
tables 9–8
overview 9–5
required authorities 9–6
utility 9–5

Progress/400 environment
creating 3–3, 3–11, 5–5
modifying 5–5

PROLODDFN utility 2–11

PROPATH value
constructing 5–11

PROSET file 8–22

PROSPORT demonstration database 1–8

PRTDEV parameter of SBMJOB command 11–14

PRTTXT parameter of SBMJOB command 11–14

Q

QCMD interface
data definition file 11–8
error handling 11–12
feedback 11–11
field size limitation 11–9
file-transfer commands 5–10
functions 11–8
multiple members 11–12
OS/400 job numbers 11–11
running RPG programs 11–10
transferring p-code 5–10

QDTAARA table 11–21

QDTAQ-ENTRY table 11–33

QSYS.LIB file system 5–3
file I/O 5–6

Queries
against joined logical files 2–22
field lists 2–23
index repositioning 2–22
performance tuning 8–21
record prefetch 2–23
tuning 2–22

QUEUE-DUPLICATE-SERVER-MESSAGES setting 8–23

QUSRSPC table 11–27

R

rcvdtaqe.p API
  coding examples 11–38
  parameters 11–36
  return values 11–38

RECEIVE DATA QUEUE API. *See*
  rcvdtaqe.p API

Receiving data, in OS/4 data queues 11–36

RECID function 2–26

Reconstruct Bad Load Records utility 9–33

Reconstructing bad load records 9–33

Record prefetch 2–23

Records
  creating 2–17, 2–26
  duplicate key 2–17
  locking 2–18
    and word indexes 2–36
  wait time for updates and deletes 2–21

Recovering
  corrupted word indexes 2–33
  databases 8–4, 8–6
    journal entries 8–6
    journal receiver 8–6

Relative paths 5–3

Reserved words, Progress/400 8–30

Retaining Progress attributes 8–28

RETRIEVE DATA AREA API. *See*
  rtvdara.p API

RETRIEVE USER SPACE API. *See*
  rtvuspc.p API

Retrieving data
  from OS/4 data areas 11–24
  from OS/4 user spaces 11–30

REVOKE statement 2–27

RMVSCHE utility 10–23
  parameters 10–23

ROOT file system 5–2
  absolute and relative paths 5–3
  accessing QSYS.LIB files 5–3
  file I/O 5–6

ROWID function 2–26

RPG programs 11–10

RPLDCTWBT utility 10–27
  parameters 10–27

RQSDTA parameter of SBMJOB command
  11–14

RTGDTA parameter of SBMJOB command
  11–14

rtvdara.p API 11–24
  coding example 11–26
  parameters 11–24
  return values 11–25

rtvuspc.p API 11–30
  coding example 11–32
  parameters 11–30
  return values 11–31

Running
  Progress applications on the AS/400
    1–10
  server utilities 10–3
  TCP/IP 4–4

S

-S connection parameter 4–14

SBMJOB command 11–8
  executing 11–10

SBS argument to -Dsrv parameter 4–9

Schema holder 1–7, 3–5, 3–13
  code page 8–12
  creating 3–7, 3–15, 9–2
  DDM 2–10
  deleting 9–5
  deploying 3–9, 3–18
  Progress database for 3–5, 3–13
  security 1–11
  synchronizing 9–3, 9–7

Schema image 1–7
  creating 5–4
  DUPPRODB 5–5

Schema server. *See* Server schema

Security 1–11
  application 2–27
  application security 1–11
  AppServer 8–4
  database 8–2
  implementation by Progress/400 8–3
  schema holder 1–11
  user permission file 2–28

Selection-by-server, connection parameter 4–9

SEND DATA QUEUE API. *See* snddtaqe.p API

Sending data
  to OS/4 data queues 11–33
  using LOOKUP and SUBSTRING functions 11–35

Sequences
  adding 9–17
  deleting 9–18
  dumping current values 9–28
  dumping definitions 9–27
  loading current values 9–33
  modifying 9–18

Server Name (-Sn) connection parameter 4–15

Server schema 1–5
  creating empty 3–3, 3–11
  CRTSCHIMG 5–5
  files 10–20
  objects 10–21
  P__ files 1–5

Server utilities. *See* Utilities

Servers, code pages 8–10

Service Name (-S) connection parameter 4–14

Setting up word indexes 2–29

Settings file (PROSET) 8–22

SETUSERID function 2–27

SHARE-LOCK state 2–18
  access conflicts 2–20, 2–22

Single-user connection mode 4–18

SMBJOB command
  parameters 11–13

-Sn connection parameter 4–15

SNA protocol 4–17
  overview 4–5
  supported routers 4–5

snddtaqe.p API 11–33
  coding examples 11–38
  parameters 11–33
  return values 11–35

Sort order. *See* Collation sequence

SPORTS2000 demonstration database 1–8

SQL
  maintaining data definitions 3–18

SQL NULL support
  DUPPRODB B–3

SQL, maintaining data definitions 3–10

Startup parameters
clients 5–12
Message Buffer Size (-Mm) 4–18, 8–22

Stopping TCP/IP brokers 4–5

Stored procedures 11–15
adding 9–19
adding a parameter 9–21
argument data types 11–17
contents of 11–16
deleting 9–21
deleting a parameter 9–24
modifying 9–20
modifying a parameter 9–23
output parameter values 11–19
programming restrictions 11–20
return codes 11–18
running 11–18

Stream files, defined 5–2

STRJRNPF command 8–7, 8–8

STRPROCLI utility 5–12, 10–35
parameters 10–35

STRPRONET utility 10–32
parameters 10–32

STRWISPRC utility 10–27
parameters 10–28

SWS parameter of SBMJOB command 11–15

Synchronize Progress/400 Client utility 9–3

Synchronizing
client 9–33
client schema holder 9–7
schema holder 9–3

SYNSCHIMG utility 10–25

Syntax notation xxi

SYSLIBL parameter of SBMJOB command 11–14

System administration 8–1

T

Table properties 9–7, 9–9, 9–10, 9–11, 9–15, 9–16, 9–17, 9–18

Tables
adding 9–7
deleting 9–9
freezing and unfreezing 9–35
modifying 9–8
virtual 2–5

TCP/IP brokers 4–3
managing 4–5

TCP/IP protocol 4–16, 4–17
overview 4–3
running 4–4

Test environments
creating 8–31

Time data type 2–11

Timestamp data type 2–11

Transaction control 8–4
See also Commitment control
and word indexes 2–31
client-based 8–5
connection parameter 4–9, 8–5

Transactions
committing 2–16
locking records 2–16

TRANSCTL argument to -Dsrv parameter 4–9, 8–5

Transferring p-code to the AS/400 5–10

Triggers, database 2–5
and word indexes 2–31

Troubleshooting database connections 4–19

Tuning DataServer performance 8–21
See also Performance tuning

Tuning queries 2–22

Typographical conventions xx

U

-U connection parameter 4–14

Unfreezing tables 9–35

Unique indexes
  unknown value restriction B–2

Unique names for database objects 2–4

Unknown values 2–14
  index restriction B–2
  Progress/400 Support B–2, B–3

UPCASE table 8–18

Updating a lock table 2–19

UPDPROWBT utility 10–28
  parameters 10–29

UPDWISTRG utility 10–30
  parameters 10–30

Upgrading to current DataServer version 1–10

Uppercase collation table 8–18

Usage guidelines
  DataServer 1–9
  documentation 1–11

USE-CHGPF setting 8–25

User ID (-U) connection parameter 4–14, 8–3

User ID, defining at startup 2–28

USER parameter of SBMJOB command 11–14

User permission file 2–28

User profile 8–3

User spaces, OS/400 11–27
  changing data 11–28
  coding example 11–32
  QUSRSPC table 11–27
  retrieving data 11–30

User-defined collation tables 8–17
  conversion tables 8–21
  implementing 8–18

USERID function 2–27

Utilities 1–8, 10–3
  CHGPRODCT 3–4, 10–8
  CHGPRODFT 10–11
  client 9–1
  Create DB2/400 DataServer Schema 9–2
  Create Incremental .df Files 9–28
  CRTPROLKT 2–19, 10–13
  CRTSCHIMG 5–5, 10–13
  CVTPRODCT 10–14
  CVTSRVSCH 10–15
  Data Administration tool 9–2
  Delete DB2/400 DataServer Schema 9–5
  DMPRODTA 10–21, 10–22
  Dump Data & Definitions 9–25
  Dump Sequences Current Values 9–28
  Dump Sequences Definitions 9–27
  Dump Table Contents 9–26
  DUPPRODB 3–3, 3–11, 5–5, 10–15
  Edit DB2/400 Connection Information 9–4
  ENDPRONET 4–5, 10–31
  ENDWISPRC 10–26
  GENWRDIDX 10–26
  Load Data & Definitions 9–30
  Load Sequences Current Values 9–33
  Load Table Contents 9–32
  MNGPRONET 4–5, 10–32
  Native 4GL Client 10–34
  networking 10–31
  online help 10–5
  Progress/400 Data DIctionary 9–5
  Progress/400 Synchronize 9–33
  PROLODDFN 2–11
  Reconstruct Bad Load Records 9–33
  RMVSCHE 10–23
  RPLDCTWBT 10–27
  STRPROCLI 5–12, 10–35

Utilities (continued)
STRPRONET 10–32
STRWISPRC 10–27
Synchronize Progress/400 Client 9–3
SYNSCHIMG 10–25
UPDPROWBT 10–28
UPDWISTRG 10–30
Word index 10–26
WRKPROJOB 8–2, 10–33

V

Validation expression 9–13

Views 2–5

Virtual tables 2–5
dumping definitions 9–26

W

Web-based architecture 1–2

Windows
operating-system code page 8–11
stream-file code page 8–12

Word breaks 2–30

Word Index Support Processor 2–32

Word Index utilities 10–26

Word Index Work Library 2–32

Word indexes 2–28
coding issues 2–34
corrupted 2–33
existing 2–29
extent fields 2–35
maintaining 2–31
new 2–30
record locking 2–36
recovery procedures 2–33
restrictions 2–29
setting up 2–29
transaction control 2–31
triggers 2–31
WHERE clause AND keyword 2–34
word-break rules, changing 2–30
Work library 2–32

WRKJRNA command 8–7

WRKPROJOB utility 8–2, 10–33
parameters 10–34

WRTSTMF command 5–10

Z

Zoned decimal data type 2–10