Technical Standard

DRDA, Version 2, Volume 1: Distributed Relational Database Architecture (DRDA)

[This page intentionally left blank]

Open Group Technical Standard

DRDA, Version 2, Volume 1:

Distributed Relational Database Architecture (DRDA)

The Open Group

Open Group Technical Standard

DRDA, Version 2, Volume 1: Distributed Relational Database Architecture (DRDA)

Document Number: C911

Published in the U.K. by The Open Group, December 1999.

Any comments relating to the material contained in this document may be submitted to:

The Open Group
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

OGSpecs@opengroup.org

Contents

Chapter 1 The DRDA Specification **1**
1.1 The DRDA Reference 2
1.1.1 What it Means to Implement Different Levels of DRDA 2
1.1.2 What it Means to Implement DRDA Level 4 2
1.1.3 What it Means to Implement DRDA Level 3 5
1.1.4 What it Means to Implement DRDA Distributed Unit of Work 8
1.2 The FD:OCA Reference 12
1.3 DDM Reference 13

Part 1 Database Access Protocol **17**

Chapter 2 Introduction to DRDA **19**
2.1 DRDA Structure and Other Architectures 19
2.2 DRDA and SQL 19
2.3 DRDA Connection Architecture 20
2.4 Types of Distribution 20
2.5 DRDA Protocols and Functions 22

Chapter 3 Using DRDA—Overall Flows **25**
3.1 Introduction to Protocol Flows 25
3.1.1 Initialization Flows 25
3.1.2 Bind Flows 29
3.1.3 SQL Statement Execution Flows 31
3.1.4 Commit Flows 32
3.1.5 Termination Flows 34

Chapter 4 The DRDA Processing Model and Command Flows **37**
4.1 DDM and the Processing Model 38
4.2 DRDA's Relationship to DDM 39
4.3 The DRDA Processing Model 40
4.3.1 DRDA Managers 40
4.3.1.1 SNA Communications Manager 40
4.3.1.2 SNA Sync Point Communications Manager 41
4.3.1.3 TCP/IP Communications Manager 41
4.3.1.4 Agent 41
4.3.1.5 Supervisor 42
4.3.1.6 Security Manager 42
4.3.1.7 Directory 42
4.3.1.8 Dictionary 42
4.3.1.9 Resynchronization Manager 43
4.3.1.10 Sync Point Manager 43
4.3.1.11 SQL Application Manager 43

4.3.1.12 Relational Database Manager 45
4.3.1.13 CCSID Manager 46
4.3.2 The DRDA Processing Model Flow 46
4.3.3 Product-Unique Extensions 52
4.3.4 Diagnostic and Problem Determination Support in DRDA 52
4.4 DDM Commands and Replies 54
4.4.1 Accessing a Remote Relational Database Manager 54
4.4.2 DRDA Security Flows 61
4.4.2.1 Identification and Authentication Security Flows 61
4.4.3 Performing the Bind Operation and Creating a Package 67
4.4.4 Deleting an Existing Package 72
4.4.5 Performing a Rebind Operation 74
4.4.6 Activating and Processing Queries 76
4.4.6.1 Fixed-Row Protocol 78
4.4.6.2 Limited Block Protocol (No LOB Data in Answer Set) 85
4.4.6.3 Limited Block Protocol (LOB Data in Answer Set) 90
4.4.7 Executing a Bound SQL Statement 96
4.4.7.1 Executing Ordinary Bound SQL Statements 97
4.4.7.2 Invoking a Stored Procedure that Returns Result Sets 100
4.4.8 Preparing an SQL Statement 108
4.4.9 Retrieving the Data Variable Definitions of an SQL Statement 110
4.4.10 Executing a Describe Table SQL Statement 112
4.4.11 Executing a Dynamic SQL Statement 114
4.4.12 Commitment of Work in DRDA 116
4.4.12.1 Commitment of Work in a Remote Unit of Work 117
4.4.12.2 Commitment of Work in a Distributed Unit of Work 120

Chapter 5 Data Definition and Exchange 137
5.1 Use of FD:OCA 137
5.2 Use of Base and Option Sets 138
5.2.1 Basic FD:OCA Object Contained in DDM 138
5.2.2 DRDA FD:OCA Object 140
5.2.3 Early and Late Descriptors 143
5.3 Relationship of DRDA and DDM Objects and Commands 145
5.3.1 DRDA Command to Descriptor Relationship 145
5.3.2 Descriptor Classes 149
5.4 DRDA Descriptor Definitions 151
5.5 Late Descriptors 152
5.5.1 Late Array Descriptors 152
5.5.1.1 SQLCA with Data Array Description 153
5.5.1.2 Data Array Description for Multi-Row Inserts 155
5.5.2 Late Row Descriptors 155
5.5.2.1 Row Description for One Data Row 156
5.5.2.2 Row Description for One Row with SQLCA and Data 157
5.5.3 Late Group Descriptors 157
5.5.3.1 SQL Data Value Group Description 159
5.5.3.2 Overriding Output Formats 160
5.6 Early Descriptors 162

5.6.1 Initial DRDA Type Representation 162
5.6.2 Early Array Descriptors 162
5.6.2.1 SQLRSLRD Array Description 163
5.6.2.2 SQLCINRD Array Description 164
5.6.2.3 SQLSTTVRB Array Description 165
5.6.2.4 SQLCA with SQLPA Array Description 166
5.6.2.5 SQLCA with SQLDA Array Description 167
5.6.3 Early Row Descriptors 168
5.6.3.1 SQL Result Set Description 168
5.6.3.2 SQL Result Set Column Information Description 169
5.6.3.3 SQL Statement Variables Description 170
5.6.3.4 SQL Statement Row Description 171
5.6.3.5 SQL Object Name Row Description 172
5.6.3.6 SQL Number of Elements Row Description 173
5.6.3.7 SQL Privileges Area Repeating Group Row Description 174
5.6.3.8 SQL Communication Area Row Description 175
5.6.3.9 SQL Data Area Row Description 176
5.6.4 Early Group Descriptors 177
5.6.4.1 SQL Result Set Group Description 177
5.6.4.2 SQL Result Set Column Information Group Description 179
5.6.4.3 SQL Statement Variables Group Description (DDM Levels Below 6) 181
5.6.4.4 SQL Statement Variables Group Description (DDM Level 6 and Above) 183
5.6.4.5 SQL Statement Group Description 185
5.6.4.6 SQL Object Name Group Description 186
5.6.4.7 SQL Number of Elements Group Description 187
5.6.4.8 SQL Privileges Area Group Description 188
5.6.4.9 SQL Communication Area Group Description 190
5.6.4.10 SQL Communication Area Exceptions Group Description 191
5.6.4.11 SQL Data Area Group Description (DDM Levels Below 6) 192
5.6.4.12 SQL Data Area Group Description (DDM Level 6 and Above) 194
5.6.4.13 SQL User-Defined Data Group Description 196
5.6.5 Early Environmental Descriptors 197
5.6.5.1 Four-Byte Integer 201
5.6.5.2 Two-Byte Integer 202
5.6.5.3 One-Byte Integer 203
5.6.5.4 Sixteen-Byte Float 204
5.6.5.5 Eight-Byte Float 205
5.6.5.6 Four-Byte Float 206
5.6.5.7 Fixed Decimal 207
5.6.5.8 Zoned Decimal 208
5.6.5.9 Numeric Character 209
5.6.5.10 Result Set Locator 210
5.6.5.11 Eight-Byte Integer 211
5.6.5.12 Large Object Bytes Locator 212

5.6.5.13 Large Object Character Locator 213
5.6.5.14 Large Object Character DBCS Locator 214
5.6.5.15 Row Identifier 215
5.6.5.16 Date 216
5.6.5.17 Time 217
5.6.5.18 Timestamp 218
5.6.5.19 Fixed Bytes 219
5.6.5.20 Variable Bytes 220
5.6.5.21 Long Variable Bytes 221
5.6.5.22 Null-Terminated Bytes 222
5.6.5.23 Null-Terminated SBCS 223
5.6.5.24 Fixed Character SBCS 224
5.6.5.25 Variable Character SBCS 225
5.6.5.26 Long Variable Character SBCS 226
5.6.5.27 Fixed-Character DBCS (GRAPHIC) 227
5.6.5.28 Variable-Character DBCS (GRAPHIC) 228
5.6.5.29 Long Variable Character DBCS (GRAPHIC) 229
5.6.5.30 Fixed Character Mixed 230
5.6.5.31 Variable Character Mixed 231
5.6.5.32 Long Variable Character Mixed 232
5.6.5.33 Null-Terminated Mixed 233
5.6.5.34 Pascal L String Bytes 234
5.6.5.35 Pascal L String SBCS 235
5.6.5.36 Pascal L String Mixed 236
5.6.5.37 SBCS Datalink 237
5.6.5.38 Mixed-Byte Datalink 238
5.6.5.39 Large Object Bytes 239
5.6.5.40 Large Object Character SBCS 240
5.6.5.41 Large Object Character DBCS (GRAPHIC) 241
5.6.5.42 Large Object Character Mixed 242
5.6.6 Late Environmental Descriptors 243
5.7 FD:OCA Meta Data Summary 245
5.7.1 Overriding Descriptors to Handle Problem Data 250
5.7.1.1 Overriding Everything 250
5.7.1.2 Overriding Some User Data 252
5.7.1.3 Assigning LIDs to O Triplets 252
5.7.2 MDD Materialization Rules 253
5.7.3 Error Checking and Reporting for Descriptors 254
5.7.3.1 General Errors 254
5.7.3.2 MDD Errors 254
5.7.3.3 SDA Errors 254
5.7.3.4 GDA / CPT Errors 255
5.7.3.5 RLO Errors 255
5.8 DRDA Examples 256
5.8.1 Environmental Description Objects 256
5.8.1.1 Early Environmental Descriptors 256
5.8.1.2 Early Data Unit Descriptors 258
5.8.1.3 Late Data Unit Descriptors 259

5.8.2 Command Execution Examples 259
5.8.2.1 EXECUTE IMMEDIATE 260
5.8.2.2 Open Query Statement 261
5.8.2.3 Insert (Multi-Row) 264
5.8.2.4 Call (Stored Procedure) 265
5.8.2.5 Call (Stored Procedure Returning Result Sets) 267

Chapter 6 Names 269
6.1 End Users 270
6.1.1 Support for End-User Names 270
6.2 RDBs 271
6.3 Tables and Views 271
6.4 Packages 273
6.4.1 Package Name 273
6.4.2 Package Consistency Token 274
6.4.3 Package Version ID 274
6.4.4 Sections 275
6.5 Stored Procedure Names 277
6.6 Synonyms and Aliases 278
6.7 Default Mechanisms for Standardized Object Names 278
6.8 Target Program 279

Chapter 7 DRDA Rules 281
7.1 Connection Allocation (CA Rules) 281
7.2 Mapping of RMs to SQLSTATEs (CD Rules) 283
7.3 Connection Failure (CF Rules) 283
7.4 Commit/Rollback Processing (CR Rules) 284
7.5 Connection Usage (CU Rules) 287
7.6 Conversion of Data Types (DC Rules) 288
7.7 Data Representation Transformation (DT Rules) 290
7.8 RDB-Initiated Rollback (IR Rules) 294
7.9 Optionality (OC Rules) 295
7.10 Program Binding (PB Rules) 296
7.11 Security (SE Rules) 299
7.12 SQL Section Number Assignment (SN Rules) 300
7.13 Stored Procedures (SP Rules) 301
7.14 SET Statement (ST Rules) 302
7.15 Serviceability (SV Rules) 303
7.16 Update Control (UP Rules) 305
7.17 Passing Warnings to the Application Requester (WN Rules) 306
7.18 Names 307
7.18.1 End-User Names (EUN Rules) 307
7.18.2 SQL Object Names (ON Rules) 307
7.18.3 Relational Database Names (RN Rules) 307
7.18.4 Target Program Names (TPN Rules) 308
7.19 Query Processing 310
7.19.1 Blocking 310
7.19.1.1 Block Formats (BF Rules) 310

7.19.1.2 Block Size (BS Rules) ... 313
7.19.1.3 Chaining (CH Rules) ... 314
7.19.2 Query Data Transfer Protocols (QP Rules) ... 316
7.19.3 Query Data or Result Set Transfer (QT Rules) ... 318
7.19.4 Additional Query and Result Set Termination Rules ... 319
7.19.4.1 Rules for OPNQRY, CNTQRY, CLSQRY, and EXCSQLSTT ... 320
7.19.4.2 Rules for FETCH ... 325
7.19.4.3 Rules for CLOSE ... 329

Chapter 8 SQLSTATE Usage ... 331
8.1 DRDA Reply Messages and SQLSTATE Mappings ... 331
8.2 SQLSTATEs that DRDA References ... 333

Part 2 Environmental Support ... 339

Chapter 9 Environmental Support ... 341
9.1 DDM Communications Model and Network Protocol Support ... 341
9.2 Accounting ... 342
9.3 Transaction Processing ... 342

Chapter 10 Security ... 343
10.1 DCE Security Mechanisms with GSS-API ... 343
10.2 Userid-Related Security Mechanisms ... 345
10.2.1 Userid and Password Security Mechanism ... 345
10.2.2 Userid, Password, and New Password Security Mechanism ... 346
10.2.3 Userid-Only Security Mechanism ... 347
10.2.4 Userid and Password Substitute Security Mechanism ... 348
10.2.5 Userid and Encrypted Password Security Mechanism ... 349

Chapter 11 Problem Determination ... 351
11.1 Network Management Tools and Techniques ... 351
11.1.1 Standard Focal Point Messages ... 351
11.1.2 Focal Point ... 351
11.1.3 Correlation ... 352
11.2 DRDA Required Problem Determination and Isolation Enhancements ... 353
11.2.1 Correlation Displays ... 353
11.2.2 DRDA Diagnostic Information Collection and Correlation ... 353
11.2.2.1 Data Collection ... 353
11.2.2.2 Correlation Between Focal Point Messages and Supporting Data ... 353
11.3 Generic Focal Point Messages and Message Models ... 354
11.3.1 When to Generate Alerts ... 354
11.3.2 Alerts and Alert Structure ... 354
11.3.2.1 Alert Implementation Basics ... 354
11.3.3 Error Condition to Alert Model Mapping ... 355
11.3.3.1 Specific Alert to DDM Reply Message Mapping ... 355
11.3.3.2 Additional Alerts at the Application Requester ... 357

11.3.3.3 DRDA-Defined Alert Models ... 357
11.3.4 Alert Example ... 375
11.3.4.1 Major Vector / Subvector / Subfield Construction ... 375

Part 3 Network Protocols ... 383

Chapter 12 SNA ... 385
12.1 SNA and the DDM Communications Model ... 385
12.2 What You Need to Know About SNA and LU 6.2 ... 386
12.3 LU 6.2 ... 387
12.4 LU 6.2 Verb Categories ... 388
12.5 LU 6.2 Product-Support Subsetting ... 389
12.6 LU 6.2 Base and Option Sets ... 390
12.6.1 Base Set Functions ... 390
12.6.1.1 Basic Conversation Verb Category ... 390
12.6.1.2 Type-Independent Verb Category ... 390
12.6.2 Option Set Functions ... 390
12.6.2.1 Basic Conversation Verb Category ... 390
12.6.2.2 Type-Independent Verb Category ... 391
12.7 LU 6.2 and DRDA ... 392
12.7.1 Initializing a Conversation ... 392
12.7.1.1 LU 6.2 Verbs that the Application Requester Uses ... 392
12.7.1.2 LU 6.2 Verbs that the Application Server Uses ... 394
12.7.1.3 Initialization Flows ... 395
12.7.2 Processing a DRDA Request ... 400
12.7.2.1 LU 6.2 Verbs that the Application Requester Uses ... 401
12.7.2.2 LU 6.2 Verbs that the Application Server Uses ... 401
12.7.2.3 Bind Flows ... 402
12.7.2.4 SQL Statement Execution Flows ... 405
12.7.3 Terminating a Conversation ... 407
12.7.3.1 LU 6.2 Verbs that the Application Requester Uses ... 408
12.7.3.2 LU 6.2 Verbs that the Application Server Uses ... 408
12.7.3.3 Termination Flow—SYNC_LEVEL(NONE) Conversation ... 409
12.7.3.4 Termination Flow—SYNC_LEVEL(SYNCPT) Conversation ... 409
12.7.4 Commit Flows on SYNC_LEVEL(NONE) Conversations ... 410
12.7.5 Rollback Flows on SYNC_LEVEL(NONE) Conversations ... 411
12.7.6 Commit Flows on SYNC_LEVEL(SYNCPT) Conversations ... 411
12.7.7 Rollback Flows on SYNC_LEVEL(SYNCPT) Conversations ... 412
12.7.8 Handling Conversation Failures ... 413
12.7.9 Managing Conversations Using Distributed Unit of Work ... 413
12.8 SNA Environment Usage in DRDA ... 414
12.8.1 Problem Determination in SNA Environments ... 414
12.8.1.1 LUWID ... 414
12.8.1.2 DRDA LUWID and Correlation of Diagnostic Information ... 414
12.8.1.3 Data Collection ... 415
12.8.1.4 Alerts and Supporting Data in SNA Environments ... 415
12.8.2 Rules Usage for SNA Environments ... 415
12.8.2.1 LU 6.2 Usage of Connection Allocation Rules ... 415

12.8.2.2 LU 6.2 Usage of Commit/Rollback Processing Rules 416
12.8.2.3 LU 6.2 Usage of Security (SE Rules) 416
12.8.2.4 LU 6.2 Usage of Serviceability Rules 417
12.8.2.5 LU 6.2 Usage of Names 417
12.8.3 Transaction Program Names 419

Chapter 13 TCP/IP 421
13.1 TCP/IP and the DDM Communications Model 421
13.2 What You Need to Know About TCP/IP 421
13.3 TCP/IP 422
13.3.1 Transport Control Protocol (TCP) 422
13.3.2 Application Services 423
13.4 Sockets Interface 423
13.5 TCP/IP and DRDA 424
13.5.1 Initializing a Connection 424
13.5.1.1 Initialization Flows 424
13.5.2 Processing a DRDA Request 426
13.5.2.1 Bind Flows 426
13.5.2.2 SQL Statement Execution Flows 427
13.5.3 Terminating a Connection 428
13.5.4 Commit Flows 429
13.5.4.1 Remote Unit of Work 429
13.5.4.2 Distributed Unit of Work Using DDM Sync Point Manager 430
13.5.5 Handling Connection Failures 432
13.6 TCP/IP Environment Usage in DRDA 433
13.6.1 Problem Determination in TCP/IP Environments 433
13.6.1.1 Standard Focal Point Messages 433
13.6.1.2 Focal Point Support 433
13.6.1.3 Correlation and Correlation Display 433
13.6.2 Rules Usage for TCP/IP Environments 434
13.6.2.1 TCP/IP Usage of Connection Allocation Rules 434
13.6.2.2 TCP/IP Usage of Commit/Rollback Processing Rules 434
13.6.2.3 TCP/IP Usage of Security (SE Rules) 435
13.6.2.4 TCP/IP Usage of Serviceability Rules 435
13.6.2.5 TCP/IP Usage of Relational Database Names Rules 435
13.6.2.6 TCP/IP Usage of PORT for DRDA Service Rules 435
13.6.3 Service Names 436

Appendix A Building Statement-Level SQLCAs for Multi-Row Fetches 437

Appendix B DDM Managers, Commands, and Reply Messages 441
B.1 DDM Manager Relationship to DRDA Functions 441
B.2 DDM Commands and Reply Messages 442

Glossary 507

Index **515**

List of Figures

2-1 Degrees of Distribution of Database Function 21
2-2 DRDA Network 23
2-3 DRDA Network Implementation Example 23
3-1 Logical Flow: Initialization Flows with SNA Security 27
3-2 Logical Flow: Initialization Flows with DCE Security 28
3-3 Logical Flow: Bind Flows 30
3-4 Logical Flow: SQL Statement Execution Flows 32
3-5 Logical Flow: DRDA Two-Phase Commit 33
3-6 Logical Flow: DRDA One-Phase Commit Using DDM Commands 34
3-7 Logical Flow: DRDA Two-Phase Commit Termination Flows Using DDM Commands 35
3-8 Logical Flow: SNA Termination Flows on Protected Conversations 35
4-1 DRDA Processing Model 47
4-2 DRDA Flows to Establish a Connection to a Remote Database Manager 55
4-3 DCE Security Flow 62
4-4 Password Encryption or Substitution Flow 65
4-5 DRDA Flows: Binding and/or Package Creation (Part 1) 67
4-6 DRDA Flows: Binding and/or Package Creation (Part 2) 68
4-7 DRDA Flows: Dropping an Existing Package 72
4-8 DRDA Flows: Rebinding an Existing Package 74
4-9 DRDA Flows: Fixed-Row Protocol Query Processing (Part 1) 78
4-10 DRDA Flows: Fixed-Row Protocol Query Processing (Part 2) 79
4-11 DRDA Flows: Limited Block Protocol Query Processing (No LOB Data) 85
4-12 Limited Block Protocol (with LOB Data, rtnextall) (Part 1) 90
4-13 Limited Block Protocol (with LOB Data, rtnextall) (Part 2) 91
4-14 Limited Block Protocol Query Processing (with LOB Data, rtnextrow) 94
4-15 DRDA Flows: Executing a Bound SQL Statement 97
4-16 DRDA Flows: Executing a Stored Procedure (Part 1) 101
4-17 DRDA Flows: Executing a Stored Procedure (Part 2) 102
4-18 DRDA Flows: Preparing an SQL Statement 108
4-19 DRDA Flows: Describing a Bound SQL Statement 110
4-20 DRDA Flows: Describing a Table 112
4-21 DRDA Flows: Immediate Execution of SQL Work 114
4-22 DRDA Flows: Commit a Remote Unit of Work 117
4-23 DRDA Flows: Executing a Bound SQL CALL Statement 119
4-24 DRDA Sample Configuration 121
4-25 DRDA RDBUPDRM Example Flow 122
4-26 Single RDB Update at a DRDA Remote Unit of Work AS 124
4-27 Single RDB Update Using Distributed Unit of Work 127

4-28 Multi-Relational Database Update 131
4-29 RDB at AS1 Initiates Rollback 135
4-30 RDB at AS2 Initiates Rollback 135
4-31 RDB at AS3 Initiates Rollback 136
5-1 Basic FD:OCA Object 138
5-2 Basic FD:OCA Object (DDM Level 6) 139
5-3 Conceptual View of a DRDA FD:OCA Object 140
5-4 Conceptual View of a DRDA FD:OCA Object with LOB Columns 142
5-5 SQLDTARD Array Descriptor (SQLCAs and Data as Reply Data) 153
5-6 SQLDTAMRW Array Descriptor (Multi-Row Insert Data) 155
5-7 SQLDTA Row Descriptor 156
5-8 SQLCADTA Row Descriptor (One Row with SQLCA and Associated Data) 157
5-9 SQLDTAGRP Group Descriptor (Field Specs for One Row of SQL Data) 159
5-10 SQLRSLRD Array Descriptor 163
5-11 SQLCINRD Array Descriptor 164
5-12 SQLSTTVRB Array Descriptor 165
5-13 SQLPARD Array Descriptor (SQLCA Followed by an SQLPA as Reply Data) 166
5-14 SQLDARD Array Descriptor (SQLCA Followed by an SQLDA as Reply Data) 167
5-15 SQLRSROW Row Descriptor (Information for One Result Set) 168
5-16 SQLCIROW Row Descriptor (Information for One Column) 169
5-17 SQLVRBROW Row Descriptor (All Information for one Variable) 170
5-18 SQLSTT Row Descriptor (One SQL Statement) 171
5-19 SQLOBJNAM Row Descriptor (One SQL Object Name) 172
5-20 SQLNUMROW Row Descriptor 173
5-21 SQLPAROW SQLPA Repeating Group Row Descriptor 174
5-22 SQLCARD Row Descriptor (SQLCA as Presented on the Link) 175
5-23 SQLDAROW Row Descriptor (SQLDA Describing Variables during Bind) 176
5-24 SQLRSGRP Group Descriptor (Information for One Result Set) 177
5-25 SQLCIGRP Group Descriptor (Information for One Column) 179
5-26 SQLVRBGRP Group Descriptor (Info. for One Variable) (DDM Levels Below 6) 181
5-27 SQLVRBGRP Group Descriptor (Info. for One Variable) (DDM Level 6 and Above) 183
5-28 SQLSTTGRP Group Descriptor (One SQL Statement) 185
5-29 SQLOBJGRP Group Descriptor (One SQL Object Name) 186
5-30 SQLNUMGRP Group Descriptor 187
5-31 SQLPAGRP Group Descriptor 188
5-32 SQLCAGRP Group Descriptor (Nullable) 190
5-33 SQLCAXGRP Group Descriptor (Nullable) 191
5-34 SQLDAGRP Group Descriptor (Info. for One Column)

(DDM Levels Below 6) 192
5-35 SQLDAGRP Group Descriptor (Info. for One Column) (DDM Level 6 and Above) 194
5-36 SQLUDTGRP Group Descriptor (UDT Information for One Column) 196
5-37 DRDA Type X‘32,33’ SQL Type 448,449 Variable Character SBCS 197
5-38 DRDA Type X‘02,03’ SQL Type 496,497 INTEGER 201
5-39 DRDA Type X‘04,05’ SQL Type 500,501 SMALL INTEGER 202
5-40 DRDA Type X‘06,07’ SQL Type n/a,n/a 203
5-41 DRDA Type X‘08,09’ SQL Type 480,481 FLOAT (16) 204
5-42 DRDA Type X‘0A,0B’ SQL Type 480,481 FLOAT (8) 205
5-43 DRDA Type X‘0C,0D’ SQL Type 480,481 FLOAT (4) 206
5-44 DRDA Type X‘0E,0F’ SQL Type 484,485 FIXED DECIMAL 207
5-45 DRDA Type X‘10,11’ SQL Type 488,489 ZONED DECIMAL 208
5-46 DRDA Type X‘12,13’ SQL Type 504,505 NUMERIC CHARACTER 209
5-47 DRDA Type X‘14,15’ SQL Type 972,973 RESULT SET LOCATOR 210
5-48 DRDA Type X‘16,17’ SQL Type 492,493 EIGHT-BYTE INTEGER 211
5-49 DRDA Type X‘18,19’ SQL Type 960,961 LARGE OBJECT BYTES LOCATOR 212
5-50 DRDA Type X‘1A,1B’ SQL Type 964,965 LARGE OBJ. CHAR. SBCS LOCATOR 213
5-51 DRDA Type X‘1C,1D’ SQL Type 968,969 LARGE OBJ. CHAR. DBCS LOCATOR 214
5-52 DRDA Type X‘1E,1F’ SQL Type 904,905 ROW IDENTIFIER 215
5-53 DRDA Type X‘20,21’ SQL Type 384,385 DATE 216
5-54 DRDA Type X‘22,23’ SQL Type 388,389 TIME 217
5-55 DRDA Type X‘24,25’ SQL Type 392,393 TIMESTAMP 218
5-56 DRDA Type X‘26,27’ SQL Type 452,453 FIXED BYTES 219
5-57 DRDA Type X‘28,29’ SQL Type 448,449 VARIABLE BYTES 220
5-58 DRDA Type X‘2A,2B’ SQL Type 456,457 LONG VAR BYTES 221
5-59 DRDA Type X‘2C,2D’ SQL Type 460,461 NULL-TERMINATED BYTES 222
5-60 DRDA Type X‘2E,2F’ SQL Type 460,461 NULL-TERMINATED SBCS 223
5-61 DRDA Type X‘30,31’ SQL Type 452,453 FIXED CHARACTER SBCS 224
5-62 DRDA Type X‘32,33’ SQL Type 448,449 VARIABLE CHARACTER SBCS 225
5-63 DRDA Type X‘34,35’ SQL Type 456,457 LONG VAR CHARACTER SBCS 226
5-64 DRDA Type X‘36,37’ SQL Type 468,469 FIXED CHARACTER DBCS 227
5-65 DRDA Type X‘38,39’ SQL Type 464,465 VARIABLE CHARACTER DBCS 228

5-66 DRDA Type X‘3A,3B’ SQL Type 472,473 LONG VAR CHARACTER DBCS 229
5-67 DRDA Type X‘3C,3D’ SQL Type 452,453 FIXED CHARACTER MIXED 230
5-68 DRDA Type X‘3E,3F’ SQL Type 448,449 VARIABLE CHARACTER MIXED 231
5-69 DRDA Type X‘40,41’ SQL Type 456,457 LONG VARIABLE CHARACTER MIXED 232
5-70 DRDA Type X‘42,43’ SQL Type 460,461 NULL-TERMINATED MIXED 233
5-71 DRDA Type X‘44,45’ SQL Type 476,477 PASCAL L STRING BYTES 234
5-72 DRDA Type X‘46,47’ SQL Type 476,477 PASCAL L STRING SBCS 235
5-73 DRDA Type X‘48,49’ SQL Type 476,477 PASCAL L STRING MIXED 236
5-74 DRDA Type X‘4C,4D’ SQL Type 396,397 SBCS DATALINK 237
5-75 DRDA Type X‘4E,4F’ SQL Type 396,397 MIXED-BYTE DATALINK 238
5-76 DRDA Type X‘C8,C9’ SQL Type 404,405 LARGE OBJECT BYTES 239
5-77 DRDA Type X‘CA,CB’ SQL Type 408,409 LARGE OBJECT CHAR. SBCS 240
5-78 DRDA Type X‘CC,CD’ SQL Type 412,413 LARGE OBJECT CHAR. DBCS 241
5-79 DRDA Type X‘CE,CF’ SQL Type 408,409 LARGE OBJECT CHAR. MIXED 242
5-80 DRDA Type X‘30’, SQL Type 468, MDD Override Example— Base 243
5-81 DRDA Type X‘30’, SQL Type 468, MDD Override Example— Override 244
10-1 Using GSS-API to Call DCE-Based Security Flows in DRDA 343
10-2 Userid and Password Authentication Flows 345
10-3 Userid, Password, and New Password Authentication Flows 346
10-4 Userid and Password Authentication Flows 347
10-5 Userid and Password Substitute Authentication Flows 348
10-6 Userid and Encrypted Password Authentication Flows 349
11-1 Summary of Required Subvectors and Subfields 355
11-2 Subfield X‘85’ for Failure Causes Code Point X‘F0A3’ 359
11-3 Subfield X‘85’s for Actions Code Point X‘32D1’ 360
11-4 Subfield X‘85’ for Actions Code Point X‘32A0’ 360
11-5 Subfield X‘85’s for Failure Causes Code Point X‘F0C0’ 372
11-6 Major Vector / Subvector / Subfield Construction 375
11-7 Alert Example for AGNPRMRM with Severity Code of 64 (Part 1) 376
11-8 Alert Example for AGNPRMRM with Severity Code of 64 (Part 2) 377
11-9 Alert Example for AGNPRMRM with Severity Code of 64

(Part 3) 378
12-1 DRDA Initialization Flows with LU 6.2 Security (Part 1) 396
12-2 DRDA Initialization Flows with LU 6.2 Security (Part 2) 397
12-3 DRDA Initialization Flows with DCE Security (Part 1) 398
12-4 DRDA Initialization Flows with DCE Security (Part 2) 399
12-5 DRDA Initialization Flows with DCE Security (Part 3) 400
12-6 DRDA Bind Flows (Part 1) 403
12-7 DRDA Bind Flows (Part 2) 404
12-8 DRDA Bind Flows (Part 3) 405
12-9 DRDA SQL Statement Execution Flows (Part 1) 406
12-10 DRDA SQL Statement Execution Flows (Part 2) 407
12-11 Actual Flow: Termination Flows on SYNC_LEVEL(NONE) Conversation 409
12-12 Actual Flow: Termination Flows on SYNC_LEVEL(SYNCPT) Conversation 410
12-13 Commit Flow for a SYNC_LEVEL(NONE) Conversation 411
12-14 Actual Flow: Commit Flow on a SYNC_LEVEL(SYNCPT) Conversation 412
12-15 Actual Flow: Backout Flow on a SYNC_LEVEL(SYNCPT) Conversation 412
13-1 TCP/IP Components 422
13-2 DRDA Initialization Flows on TCP/IP with DCE Security 425
13-3 DRDA Bind Flows on TCP/IP 427
13-4 DRDA SQL Statement Execution Flows on TCP/IP 428
13-5 DRDA Termination Flows on TCP/IP 429
13-6 DRDA Server Abnormal Termination Flows on TCP/IP 429
13-7 DRDA Commit Flows on TCP/IP 430
13-8 TCP/IP Distributed Unit of Work Commit Flow 431
13-9 TCP/IP Distributed Unit of Work Rollback Flow 432

List of Tables

1-1 DDM Modeling and Description Terms 13
1-2 DDM Terms of Interest to DRDA Implementers 14
1-3 DDM Command Objects Used by DRDA 15
1-4 DDM Reply Data Objects Used by DRDA 16
4-1 DDM Commands Used in DRDA Flows 45
4-2 Access by the Minimum MGRLVLLS Parameter of EXCSAT and EXCSATRD 57
4-3 Security Mechanism to *secmec* Value Mapping 62
5-1 Data Objects, Descriptors, and Contents for DRDA Commands 146
5-2 QRYDSC with Default Formats 161
5-3 OUTOVR with One Override Triplet 161
5-4 SQL Result Set Field Usage 178
5-5 SQL Result Set Column Information Field Usage 180
5-6 DRDA SQL Data Area Field Usage (DDM Levels Below 6) 182
5-7 DRDA SQL Data Area Field Usage (DDM Level 6 and Above) 184
5-8 DRDA SQL Privileges Area Repeating Field Usage 189

5-9 DRDA SQL Data Area Field Usage (DDM Levels Below 6) 193
5-10 DRDA SQL Data Area Field Usage (DDM Level 6 and Above) 195
5-11 DRDA SQL UDT Description Field Usage" 196
5-12 MDD References Used in Early Environmental Descriptors 245
5-13 MDD References for Early Group Data Units 247
5-14 MDD References for Early Row Descriptors 248
5-15 MDD References for Early Array Descriptors 248
5-16 MDD References Used in Late Environmental Descriptors 249
5-17 MDD References for Late Group Data Units 249
5-18 MDD References for Late Row Descriptors 249
5-19 MDD References for Late Array Descriptors 250
5-20 TYPDEFNAM and TYPDEFOVR 251
5-21 Complete set of Early Environmental Descriptors for QTDSQL370 .. 256
5-22 Complete set of Early Data Unit Descriptors 258
5-23 Complete Set of Late Data Unit Descriptors 259
5-24 STATS Sample Table 260
5-25 EXECUTE IMMEDIATE Command Data 260
5-26 EXECUTE IMMEDIATE Reply Data 260
5-27 Open Query Command Data 261
5-28 Open Query Reply Data 263
5-29 Multi-Row Insert Command Data 264
5-30 Object Data Stream Example for Execution of CALL Statement 265
5-31 Reply Data Stream Example for Execution of CALL Statement 266
5-32 Reply Data Stream Example for Summary Component of Response 267
7-1 Maximal Example for OPNQRY 314
7-2 Maximal Example for the First Block of an EXCSQLSTT that Returns Result Sets 315
7-3 AS Rules for OPNQRY, CNTQRY, CLSQRY 320
7-4 AR Rules for FETCH 325
7-5 AR Rules for CLOSE 329
8-1 DRDA Reply Messages (RMs) and Corresponding SQLSTATEs 331
11-1 Alerts Required for DDM Reply Messages 356
11-2 Additional Alerts Required at Application Requester 357
11-3 Alert Model AGNPRM 358
11-4 Alert Model BLKERR 362
11-5 Alert Model CHNVIO 363
11-6 Alert Model CMDCHK 364
11-7 Alert Model CMDVLT 365
11-8 Alert Model DSCERR 366
11-9 Alert Model GENERR 367
11-10 Alert Model PRCCNV 368
11-11 Alert Model QRYERR 369
11-12 Alert Model RDBERR 370
11-13 Alert Model RSCLMT 371
11-14 Alert Model SECVIOL 373
11-15 Alert Model SYNTAX 374
A-1 Setting of the Statement-Level SQLCA 437
B-1 DDM Manager Relationship to DRDA Level 441

B-2 ABNUOWRM Reply Message Instance Variables 443
B-3 ACCRDB Command Instance Variables 444
B-4 Reply Objects for the ACCRDB Command 444
B-5 ACCRDBRM Reply Message Instance Variables 445
B-6 ACCSEC Command Instance Variables 446
B-7 ACCSECRD Reply Object Instance Variables 446
B-8 AGNPRMRM Reply Message Instance Variables 447
B-9 BGNBND Command Instance Variables 448
B-10 Command Objects for the BGNBND Command 448
B-11 Reply Objects for the BGNBND Command 449
B-12 BGNBNDRM Reply Message Instance Variables 450
B-13 BNDSQLSTT Command Instance Variables 451
B-14 Command Objects for the BNDSQLSTT Command 451
B-15 Reply Objects for the BNDSQLSTT Command 451
B-16 CLSQRY Command Instance Variables 452
B-17 Reply Objects for the CLSQRY Command 452
B-18 CMDATHRM Reply Message Instance Variables 453
B-19 CMDCHKRM Reply Message Instance Variables 454
B-20 CMDNSPRM Reply Message Instance Variables 455
B-21 CMDVLTRM Reply Message Instance Variables 456
B-22 CMMRQSRM Reply Message Instance Variables 457
B-23 CNTQRY Command Instance Variables 458
B-24 Reply Objects for the CNTQRY Command 458
B-25 DRPPKG Command Instance Variables 459
B-26 Reply Objects for the DRPPKG Command 459
B-27 DSCINVRM Reply Message Instance Variables 460
B-28 DSCPVL Command Instance Variables 461
B-29 Command Objects for the DSCPVL Command 461
B-30 Reply Objects for the DSCPVL Command 461
B-31 DSCRDBTBL Command Instance Variables 462
B-32 Command Objects for the DSCRDBTBL Command 462
B-33 Reply Objects for the DSCRDBTBL Command 462
B-34 DSCSQLSTT Command Instance Variables 463
B-35 Reply Objects for the DSCSQLSTT Command 463
B-36 DTAMCHRM Reply Message Instance Variables 464
B-37 ENDBND Command Instance Variables 465
B-38 Reply Objects for the ENDBND Command 465
B-39 ENDQRYRM Reply Message Instance Variables 466
B-40 ENDUOWRM Reply Message Instance Variables 467
B-41 EXCSAT Command Instance Variables 468
B-42 EXCSATRD Reply Object Instance Variables 468
B-43 EXCSQLIMM Command Instance Variables 469
B-44 Command Objects for the EXCSQLIMM Command 469
B-45 Reply Objects for the EXCSQLIMM Command 469
B-46 EXCSQLSTT Command Instance Variables 470
B-47 Command Objects for the EXCSQLSTT Command 470
B-48 Reply Objects for the EXCSQLSTT Command 470
B-49 MGRDEPRM Reply Message Instance Variables 471

B-50 MGRLVLRM Reply Message Instance Variables 472
B-51 OBJNSPRM Reply Message Instance Variables 473
B-52 OPNQFLRM Reply Message Instance Variables 474
B-53 OPNQRY Command Instance Variables 475
B-54 Command Objects for the OPNQRY Command 475
B-55 Reply Objects for the OPNQRY Command 475
B-56 OPNQRYRM Reply Message Instance Variables.................................. 476
B-57 PKGBNARM Reply Message Instance Variables.................................. 477
B-58 PKGBPARM Reply Message Instance Variables 478
B-59 PRCCNVRM Reply Message Instance Variables 479
B-60 PRMNSPRM Reply Message Instance Variables 480
B-61 PRPSQLSTT Command Instance Variables 481
B-62 Command Objects for the PRPSQLSTT Command 481
B-63 Reply Objects for the PRPSQLSTT Command 481
B-64 QRYNOPRM Reply Message Instance Variables.................................. 482
B-65 QRYPOPRM Reply Message Instance Variables................................... 483
B-66 RDBACCRM Reply Message Instance Variables 484
B-67 RDBAFLRM Reply Message Instance Variables 485
B-68 RDBATHRM Reply Message Instance Variables 486
B-69 RDBCMM Command Instance Variable 487
B-70 Reply Objects for the RDBCMM Command 487
B-71 RDBNACRM Reply Message Instance Variables.................................. 488
B-72 RDBNFNRM Reply Message Instance Variables 489
B-73 RDBRLLBCK Command Instance Variables.. 490
B-74 Reply Objects for the RDBRLLBCK Command 490
B-75 RDBUPDRM Reply Message Instance Variables 491
B-76 REBIND Command Instance Variables.. 492
B-77 Command Objects for the REBIND Command 492
B-78 Reply Objects for the REBIND Command 492
B-79 RSCLMTRM Reply Message Instance Variables................................... 493
B-80 RSLSETRM Reply Message Instance Variables...................................... 494
B-81 SECCHK Command Instance Variables 495
B-82 Command Objects for the SECCHK Command 495
B-83 Reply Objects for the SECCHK Command 495
B-84 SECCHKRM Reply Message Instance Variables................................... 496
B-85 SQLERRRM Reply Message Instance Variables..................................... 497
B-86 SYNCCTL Command Instance Variables.. 498
B-87 Command Objects for SYNCCTL 498
B-88 SYNCCRD Reply Object Instance Variables 499
B-89 SYNCLOG Reply Object Instance Variables 500
B-90 SYNCRSY Command Instance Variables 501
B-91 Command Objects for SYNCRSY... 501
B-92 SYNCRRD Reply Object Instance Variables 502
B-93 Reply Objects for SYNCRRD 502
B-94 SYNTAXRM Reply Message Instance Variables 503
B-95 TRGNSPRM Reply Message Instance Variables..................................... 504
B-96 VALNSPRM Reply Message Instance Variables 505

Preface

The Open Group

The Open Group is a vendor and technology-neutral consortium which ensures that multi-vendor information technology matches the demands and needs of customers. It develops and deploys frameworks, policies, best practices, standards, and conformance programs to pursue its vision: the concept of making all technology as open and accessible as using a telephone.

The mission of The Open Group is to deliver assurance of conformance to open systems standards through the testing and certification of suppliers' products.

The Open group is committed to delivering greater business efficiency and lowering the cost and risks associated with integrating new technology across the enterprise by bringing together buyers and suppliers of information systems.

Membership of The Open Group is distributed across the world, and it includes some of the world's largest IT buyers and vendors representing both government and commercial enterprises.

More information is available on The Open Group Web Site at *http://www.opengroup.org*.

Open Group Publications

The Open Group publishes a wide range of technical documentation, the main part of which is focused on development of Technical and Product Standards and Guides, but which also includes white papers, technical studies, branding and testing documentation, and business titles. Full details and a catalog are available on The Open Group Web Site at *http://www.opengroup.org/pubs*.

- Product Standards

  A Product Standard is the name used by The Open Group for the documentation that records the precise conformance requirements (and other information) that a supplier's product must satisfy. Product Standards, published separately, refer to one or more Technical Standards.

  The ''X'' Device is used by suppliers to demonstrate that their products conform to the relevant Product Standard. By use of the Open Brand they guarantee, through the Open Brand Trademark License Agreement (TMLA), to maintain their products in conformance with the Product Standard so that the product works, will continue to work, and that any problems will be fixed by the supplier. The Open Group runs similar conformance schemes involving different trademarks and license agreements for other bodies.

- Technical Standards (formerly CAE Specifications)

  Open Group Technical Standards, along with standards from the formal standards bodies and other consortia, form the basis for our Product Standards (see above). The Technical Standards are intended to be used widely within the industry for product development and procurement purposes.

  Technical Standards are published as soon as they are developed, so enabling suppliers to proceed with development of conformant products without delay.

  Anyone developing products that implement a Technical Standard can enjoy the benefits of a single, widely supported industry standard. Where appropriate, they can demonstrate product compliance through the Open Brand.

- CAE Specifications

  CAE Specifications and Developers' Specifications published prior to January 1998 have the same status as Technical Standards (see above).

- Preliminary Specifications

  Preliminary Specifications have usually addressed an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations. There is a strong preference to develop or adopt more stable specifications as Technical Standards.

- Consortium and Technology Specifications

  The Open Group has published specifications on behalf of industry consortia. For example, it published the NMF SPIRIT procurement specifications on behalf of the Network Management Forum (now TMF). It also published Technology Specifications relating to OSF/1, DCE, OSF/Motif, and CDE.

In addition, The Open Group publishes Product Documentation. This includes product documentation—programmer's guides, user manuals, and so on—relating to the DCE, Motif, and CDE. It also includes the Single UNIX Documentation, designed for use as common product documentation for the whole industry.

Versions and Issues of Specifications

As with all *live* documents, Technical Standards and Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards compatible and those which are not:

- A new *Version* indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it *replaces* the previous publication.

- A new *Issue* indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Corrigenda

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published on The Open Group Web Site at *http://www.opengroup.org/corrigenda*.

Ordering Information

Full catalog and ordering information on all Open Group publications is available on The Open Group Web Site at *http://www.opengroup.org/pubs*.

This Document

The *Distributed Relational Database Architecture Specification* comprises three volumes:

- *Distributed Relational Database Architecture (DRDA)* (the DRDA Reference)
- *Formatted Data Object Content Architecture (FD:OCA)* (the FD:OCA Reference)
- *Distributed Data Management (DDM) Architecture* (the DDM Reference)

This volume, *Distributed Relational Database Architecture*, describes the connectivity between relational database managers that enables applications programs to access distributed relational data.

This volume describes the necessary connection between an application and a relational database management system in a distributed environment. It describes the responsibilities of these participants, and specifies when the flows should occur. It describes the formats and protocols required for distributed database management system processing. It does *not* describe an Application Programming Interface (API) for distributed database management system processing.

This reference is divided into three parts. The first part describes the database access protocols. The second part describes the environmental support that DRDA requires, which includes network support. The third part contains the specific network protocols and characteristics of the environments these protocols run in, along with how these network protocols relate to DRDA.

Note:

To understand DRDA, the reader should be familiar with the following:

- Distributed Data Management (DDM)
- Formatted Data Object Content Architecture (FD:OCA)
- Structured Query Language (SQL) and Character Data Representation Architecture (CDRA)
- At least one of the defined network protocols: Systems Network Architecture (SNA) or TCP/IP

Intended Audience

This volume is intended for relational database management systems (DBMS) development organizations. Programmers who wish to code their own connections between database management systems can use this description of DRDA as a basis for their code.

Typographic Conventions

The following typographical conventions are used throughout this document:

- **Bold** font is used for system elements that must be used literally, such as interface names and defined constants.
- *Italic* strings are used for emphasis or to identify the first instance of a word requiring definition. Italics in text also denote function names and variable values such as interface arguments.
- Normal font is used for the names of constants and literals.
- The notation **<file.h>** indicates a header file.
- The notation [EABCD] is used to identify an error value EABCD.
- Syntax, code examples, and user input in interactive examples are shown in fixed width font.
- Variables within syntax statements are shown in *italic fixed width font*.

Problem Reporting

For any problems with DRDA-based software or vendor-supplied documentation, contact the software vendor's customer service department. Comments relating to this Open Group Technical Standard, however, should be sent to the addresses provided on the copyright page.

Trademarks

Referenced Documents

These publications provide the background for understanding DRDA.

DRDA Overview

For an overview of DRDA, read:

- Open Group Technical Standard, December 1999, DRDA, Version 2, Volume 1: Distributed Relational Database Architecture (DRDA) (C911) (this document).

The DRDA Processing Model and Command Flows

These publications help the reader to understand the DDM documentation and what is needed to implement the base functions required for a DRDA product:

- Open Group Technical Standard, December 1999, DRDA, Version 2, Volume 3: Distributed Data Management (DDM) Architecture (C913).
- *Distributed Data Management Architecture General Information*, GC21-9527 (IBM).
- *Distributed Data Management Architecture Implementation Programmer's Guide*, SC21-9529 (IBM).
- *Character Data Representation Architecture Reference*, SC09-1390 (IBM).
- *Character Data Representation Architecture Registry*, SC09-1391 (IBM).

Communications, Security, Accounting, and Transaction Processing

For information about distributed transaction processing, see the following:

- CAE Specification, November 1995, Distributed Transaction Processing: The CPI-C Specification, Version 2 (ISBN: 1-85912-135-7, C419), published by The Open Group.

The following publications contain background information adequate for an in-depth understanding of DRDA's use of TCP/IP:

- *Internetworking With TCP/IP Volume I: Principles, Protocols, and Architecture*, Douglas E. Corner, Prentice Hall, Englewood Cliffs, New Jersey, 1991, SC31-6144 (IBM).
- *Internetworking With TCP/IP Volume II: Implementation and Internals*, Douglas E. Corner, Prentice Hall, Englewood Cliffs, New Jersey, 1991, SC31-6145 (IBM).
- *Internetworking With TCP/IP*, Douglas E. Corner, SC09-1302 (IBM).
- *UNIX Network Programming*, W. Richard Stevens, Prentice Hall, Englewood Cliffs, New Jersey, 1990, SC31-7193 (IBM).
- *UNIX Networking*, Kochan and Wood, Hayden Books, Indiana, 1989.
- *Introduction to IBM's Transmission Control Protocol/Internet Protocol Products for OS/2, VM, and MVS*, GC31-6080 (IBM).
- *Transmission Control Protocol*, RFC 793, Defense Advanced Research Projects Agency (DARPA).

Many IBM publications contain detailed discussions of SNA concepts and the LU 6.2 architecture. The following publications contain background information adequate for an in-depth understanding of DRDA's use of LU 6.2 functions:

- *SNA Concepts and Products*, GC30-3072 (IBM).
- *SNA Technical Overview*, GC30-3073 (IBM).
- *SNA Transaction Programmer's Reference Manual for LU Type 6.2,* GC30-3084 (IBM).
- *SNA LU 6.2 Reference: Peer Protocols*, SC31-6808 (IBM).
- *SNA Management Services: Alert Implementation Guide*, SC31-6809 (IBM).
- *SNA Format and Protocol Reference Manual: Architecture Logic For LU Type 6.2* SC30-3269 (IBM).

These are publications that contain background for DRDA's use of The Open Group's OSF DCE security. A listing of security publications is available on The Open Group website at *http://www.opengroup.org*, under publications. Many titles are available for browsing in HTML.

- CAE Specification, December 1995, Generic Security Service API (GSS-API) Base (ISBN: 1-85912-131-4, C441), published by The Open Group.
- CAE Specification, August 1997, DCE 1.1: Authentication and Security Services (C311), published by The Open Group as an electronic document (via ftp).
- *The Open Group's OSF DCE SIG Request For Comments 5.x*, GSS-API Extensions for DCE, available from The Open Group.
- *IETF Request For Comments 1508*, Generic Security Service Application Program Interface.
- *IETF Request For Comments 1510*, The Kerberos Network Authentication Service (V5).

Data Definition and Exchange

The following publications describe ISO SQL, FD:OCA, and CDRA:

- Open Group Technical Standard, December 1999, DRDA, Version 2, Volume 2: Formatted Data Object Content Architecture (FD:OCA) (C912).
- ISO / IEC 9075: 1992, Information Technology — Database Language SQL (technically identical to ANSI standard X3.135-1992).
- *Character Data Representation Architecture Reference*, SC09-1390 (IBM).
- *Character Data Representation Architecture Registry*, SC09-1391 (IBM).
- *Character Data Representation Architecture, Executive Overview*, GC09-1392 (IBM).

Other

- *ANSI/IEEE Std. 745-1985, Binary Floating Point Arithmetic*.

Chapter 1

The DRDA Specification

The *Distributed Relational Database Architecture* Specification comprises three volumes:

- *Distributed Relational Database Architecture (DRDA)* (the DRDA Reference)
- *Formatted Data Object Content Architecture (FD:OCA)* (the FD:OCA Reference)
- *Distributed Data Management (DDM) Architecture* (the DDM Reference)

DRDA is an open, published architecture that enables communication between applications and database systems on disparate platforms, whether those applications and database systems are provided by the same or different vendors and whether the platforms are the same or different hardware/software architectures. DRDA is a combination of other architectures and the environmental rules and process model for using them. The architectures that actually comprise DRDA are Distributed Data Management (DDM) and Formatted Data Object Content Architecture (FD:OCA).

The Distributed Data Management (DDM) architecture provides the overall command and reply structure used by the distributed database. Fewer than 20 commands are required to implement all of the distributed database functions for communication between the Application Requester (client) and the Application Server.

The Formatted Data Object Content Architecture (FD:OCA) provides the data definition architectural base for DRDA. Descriptors defined by DRDA provide layout and data type information for all the information routinely exchanged between the Application Requesters and Servers. A descriptor organization is defined by DRDA to allow dynamic definition of user data that flows as part of command or reply data. DRDA also specifies that the descriptors only have to flow once per answer set, regardless of the number of rows actually returned, thus minimizing data traffic on the wire.

It is recommended that the DRDA Reference be used as the main source of information and roadmap for implementing DRDA. This section describes the relationships between the above three volumes and provides the details on how they are used to develop a DRDA requester (client) or server. Overviews of DDM and FD:OCA are provided in this section and in more detail in the introductory sections of their respective volumes.

It is recommended that the introductory chapter of the DDM Reference, which describes its overall structure and basic concepts, is read either before reading Chapter 4 on page 37 or in conjunction with it. The rest of the DDM Reference should be used primarily as a reference when additional detail is needed to implement the functions and flows as defined in the DRDA Reference. Similarly, one can use the overview of FD:OCA below and the introductory section of its respective volume and only refer to the details of the FD:OCA constructs as needed during implementation.

DRDA can flow over either SNA or TCP/IP transport protocols and the details and differences in doing so are provided in the third part of the DRDA Reference. It is expected that the developer is familiar with whichever transport protocol will be supported, as that level of detail is not provided in this documentation. Even if only implementing for TCP/IP, it is recommended that the developer be familiar with the two-phase commit recovery model as described in SNA LU 6.2 since that is the model used by DRDA for either of the transport protocols.

Besides SNA and TCP/IP, DRDA also uses the following other architectures:

- Character Data Representation Architecture (CDRA)
- SNA Management Services Architecture (MSA) for problem determination support
- The Open Group Distributed Computing Environment (DCE)

For a better understanding of DRDA, the reader should have some familiarity with these architectures. (See **Referenced Documents** on page xxiv.)

Finally, DRDA is based on the Structured Query Language (SQL) but is not dependent on any particular level or dialect of it. It is not necessary to know the details of how to construct all the SQL statements, only to recognize certain types of statements and any host variables they may contain in order to map them to their DRDA equivalents.

1.1 The DRDA Reference

The DRDA Reference describes the necessary connection between an application and a relational database management system in a distributed environment. It describes the responsibilities of these participants, and specifies when the flows should occur. It describes the formats and protocols required for distributed database management system processing. It does *not* describe an Application Programming Interface (API) for distributed database management system processing.

This reference is divided into three parts. The first part describes the database access protocols. The second part describes the environmental support that DRDA requires, which includes network support. The third part contains the specific network protocols and characteristics of the environments these protocols run in, along with how these network protocols relate to DRDA.

1.1.1 What it Means to Implement Different Levels of DRDA

This version of the DRDA reference includes DRDA application-directed Remote Unit of Work (RUOW), DRDA application-directed Distributed Unit of Work (DUOW), the initial support for DRDA database-directed Distributed Unit of Work (DUOW), and the totality of SQL-related functions. It is written with the intention to allow an implementer to implement any of the DRDA functions and either Remote Unit of Work or Distributed Unit of Work types of distribution. DRDA, Version 2 adds support for an RDB implementer to provide support for database-directed Distributed Unit of Work.

1.1.2 What it Means to Implement DRDA Level 4

Describe Input

Describe input is a performance and usability enhancement to allow an application requester to obtain a description of input parameters from the RDB in a consistent format. Input parameters for dynamic SQL can be described by the characteristics of their related columns, and this column information is kept in the RDB catalog tables. Prior to DRDA Level 4, an application requester was required to do SQL statement parsing and expensive catalog lookups to determine the input parameter marker data types. With DRDA Level 4, to obtain a description of the input parameters, the Describe SQL statement command can request the RDB to return the description of input variables for a prepared SQL statement.

Describe input requires the following DDM support:

- Both Agent and SQLAM managers at Level 6.
- Support for the TYPSQLDA instance variable on the DSCSQLSTT command to request a description of the input parameters of a prepared statement.

Database-Directed Access

In database-directed requests, an application connects to a relational database management system (RDB) that can execute one or more SQL requests locally or route some or all of the SQL requests to other RDBs. The RDB determines which system manages the data referenced by the SQL statement and automatically directs the request to that system. Refer to Section 7.14 on page 302 for description on when special registers are propagated.

Database-directed access requires the following DDM support:

- Both Agent and SQLAM managers at Level 6.
- Support for the EXCSQLSET command to propagate the settings of special registers to a database server.

Two New Security Mechanisms

Two new security mechanisms are added to allow a user to be authenticated without requiring passwords to flow in the data stream as clear text.

Password Encryption Security Mechanism (PWDENC) specifies a method to encrypt the password. This mechanism authenticates the user like the userid and password mechanism, but the password is encrypted and decrypted using 56-bit DES. Diffie-Hellman public-key distribution is used to generate a shared private key. This Diffie-Hellman key and the userid are used as the DES encryption and decryption seeds.

Password Substitution Security Mechanism (PWDSBS) specifies the use of a password substitute. A password does not flow. A password substitute is generated and sent to the application server. The application server generates the password substitute and compares it with the application requester's password substitute. If equal, the user is authenticated.

Password encryption and password substitute mechanisms require the following DDM support:

- Security Manager (SECMGR) at Level 6
- New security token instance variable in the access security command (ACCSEC) and reply data (ACCSECRD)

Two New Data Types

Support for a datalink data type and an eight-byte integer data type are added. For details on eight-byte integers and datalinks, refer to the early environmental descriptors described in Chapter 5 on page 137.

Note: Datalinks extend the usefulness of relational databases by allowing SQL tables to reference non-SQL data that is more appropriately stored in other types of files. Video data, for example, may be able to be accessed much faster and more efficiently if it is stored on some file server. With the introduction of the SQL datalink data type, DRDA needs to be able to interchange this type of data between all RDBs. DRDA does not define the semantics of the contents of the datalink data type. It only provides the mechanism to pass the datalink value to and from an application requester and application server.

Datalinks and eight-byte integers require the following DDM support:

- SQLAM Manager (SECMGR) at Level 6

New Bind Option Values

In support of user-defined functions (UDFs) and stored procedures, new bind option values for package authorization rules are added. The previously supported values, OWNER and REQUESTER, are inadequate to describe the additional complexity associated with the definition and invocation of UDFs and stored procedures. Refer to the DDM Reference, PKGATHRUL for a description of the new values.

The new package authorization rules require the following DDM support:

- SQLAM Manager (SECMGR) at Level 6

Object-Oriented Extensions

The following elements of object-oriented technology have been added:

- Support for User-defined Distinct Types (UDTs)
- Support for Large Objects (LOBs)

Support for User-defined Distinct Types (UDTs) requires an SQLAM manager at Level 6 and includes the following:

- Support for a new early group, SQLUDTGRP
- Support for an enhanced SQLDAGRP which includes the SQLUDTGRP

Support for Large Objects (LOBs) requires an SQLAM manager at Level 6 and includes the following:

- Support for two new FD:OCA data types, Generalized Byte String and Generalized Character String
- Support for two new data types for eight-byte integers, allowing the manipulation of objects whose lengths are greater than 2,147,483,647 bytes:

— Eight-byte Integers

— Nullable Eight-byte Integers

- Support for two new DRDA types for row identifiers, allowing the association of the data for a large object column with the row in the base table to which it belongs:

— Row Identifier

— Nullable Row Identifier

- Support for 14 new DRDA types for LOB SQL types, allowing the manipulation of large object types:

— Large Object Bytes

— Nullable Large Object Bytes

— Large Object Character SBCS

— Nullable Large Object Character SBCS

— Large Object Character DBCS

- — Nullable Large Object Character DBCS
- — Large Object Character Mixed
- — Nullable Large Object Character Mixed
- — Large Object Bytes Locator
- — Nullable Large Object Bytes Locator
- — Large Object Character Locator
- — Nullable Large Object Character Locator
- — Large Object Character DBCS Locator
- — Nullable Large Object Character DBCS Locator

- Support in DRDA for sending and receiving the new LOB DRDA data types:
  - — SQLDTAGRP supports an FD:OCA placeholder indicator which is set on when LOB data values flow as externalized data.
  - — SQLDAGRP supports 8-byte lengths.
  - — SQLVRBGRP supports 8-byte lengths.
- Support in DDM for sending and receiving the new LOB DRDA data types:
  - — Support for EXTDTA, a new DDM object to flow externalized FD:OCA data, and support for the rules for how this data flows in the DDM data stream, as described in the FIXROWPRC and the LMTBLKPRC terms
  - — Support for the *outovropt* instance variable in an OPNQRY command
  - — Support for the *outovropt* instance variable in an EXCSQLSTT command for stored procedure calls
  - — Support for the *rtnextdta* instance variable in a CNTQRY command
  - — Support for an OUTOVR command data object for a CNTQRY command or for an EXCSQLSTT command which is not a stored procedure call
- Support in DRDA for sending and receiving the new row identifier data types.

1.1.3 What it Means to Implement DRDA Level 3

This section provides an overview of the previous functions and support that were added for DRDA, Version 1, including support for TCP/IP connections, enhanced security, stored procedures, work load balancing, and the Data Staging Area for data replication. DRDA Remote Unit of Work or DRDA Distributed Unit of Work may serve as the base for DRDA, depending on the type of distribution supported by the requester.

DRDA includes the following functions that enhance the DRDA RUOW or DRDA DUOW support:

- TCP/IP Communications manager can be supported on a DRDA RUOW base or DRDA DUOW base.
- Enhanced Security can also be supported on a DRDA RUOW base or DRDA DUOW base. Enhanced security includes additional support using Distributed Computing Environment (DCE) security mechanisms and the capability to change a password at a server.

- Enhanced Sync Point Manager with optimized two-phase commit and optional resync server support requires a DRDA DUOW base. A resync server allows an application requester to migrate resynchronization responsibilities to an application server eliminating the requirement of a recovery log at the application requester.
- Stored Procedures with result sets can be supported using a DRDA Distributed Unit of Work base.
- Enhanced Bind Options can be supported using a DRDA Distributed Unit of Work base. This allows unarchitected bind options (generic) to be sent to a server and provides a new optional package authorization rule bind option.
- Server List allows a multi-homed relational database manager server to provide work load balancing information to an application requester and can be supported using a DRDA Distributed Unit of Work base.
- Data Staging Area which is independent of the DRDA type of distribution.

It is assumed that all required functions for DRDA Remote Unit of Work or DRDA Distributed Unit of Work would be implemented as defined in this reference.

TCP/IP Communications

TCP/IP network connections requires the following DDM support:

- TCP/IP Communications Manager (CMNTCPIP) at Level 5 (see Section 4.3.1.3 on page 41 and Chapter 13 on page 421)
- Security Manager (SECMGR) at Level 5

Enhanced Security

New security mechanisms are provided to authenticate end users independent of the communications manager being used. These are in addition to extending existing mechanisms such as userid and password authentication using The Open Group's OSF DCE and the ability to change passwords for an authenticated end user. These new and enhanced security mechanisms require a new security manager level. Both the requester and server must support the enhanced security manager. Enhanced security requires the following DDM support:

- Security Manager (SECMGR) at Level 5
- Access security (ACCSEC) command and reply data (ACCSECRD) (see Section 4.4.2 on page 61 and Chapter 10 on page 343)
- Security check (SECCHK) command and reply message (SECCHKRM) (see Section 4.4.2 on page 61 and Chapter 10 on page 343)
- SECVIOL alert (see Table 11-1 on page 356 and Table 11-14 on page 373)
- Support for at least one security mechanism outside of security provided by the network (see Section 4.4.2 on page 61)

Enhanced Sync Point Manager

Distributed unit of work network connections use DDM to flow two-phase commit messages and perform resynchronizations. Refer to the DDM Sync point overview (SYNCPTOV) for a description of the enhancement. Enhanced Sync Point Manager support requires the following DDM support:

- SNA LU 6.2 Communications (CMNAPPC) at Level 3 (see Section 4.3.1.1 on page 40) or TCP/IP Communications Manager (CMNTCPIP) at Level 5 (see Section 4.3.1.3 on page 41)
- Sync Point Manager (SYNCPTMGR) at Level 5

  SNA Sync Point Manager (CMNSYNMGR) at Level 4 is mutually exclusive with Sync Point Manager at Level 5.
- Agent Resource Manager (AGENT) at Level 5

  A new level is introduced to support a new type of RQSDSS, a request with no expected reply.
- Resynchronization Manager (RSYNCMGR) at Level 5

  Initiates resynchronization to complete in doubt units of work. If RSYNCMGR at Level 5 and SYNCPTMGR at Level 5 is exchanged during the initialization of a connection, resync server support may be used on the connection to perform a two-phase commit. If supported, the application server performs logging and resynchronization on behalf of the application requester.

Stored Procedures

Stored procedures with multi-row result sets require the following DDM support:

- SQL Application Manager (SQLAM) at Level 5
- Host variables in SQLDTARD that are associated with a CALL (see Section 4.4.7.1 on page 97)
- Handling commit and rollback in a stored procedure in a remote unit of work (see **Commit and Rollback Scenarios** on page 122)
- Handling commit and rollback in a stored procedure in a distributed unit of work (see **Commit and Rollback Scenarios** on page 122)
- Receipt of prcnamon EXCSQLSTT and the semantics of receiving it (see Section 4.4.7.1 on page 97)
- Result sets (see Section 4.4.7.2 on page 100)

Server List

The Server List is an option on the access RDB reply message. It contains a weighted list of network addresses that can be used to access the RDB. The list can be used by the requester to work load balance future connections. Details of the server list and examples are in the DDM references. Server List requires the following DDM support:

- SQL Application Manager (SQLAM) at Level 5
- Support of srvlston ACCRDBRM and the semantics of sending and processing it (see Section 4.4.3 on page 67)

Enhanced Bind Options

Package Authorization Rules bind option and generic bind options consist of the following DDM support:

- SQL Application Manager (SQLAM) at Level 5
- Support of pkgathrulon BGNBND and the semantics of sending and processing it (see Section 4.4.3 on page 67)
- Support of bndopt object on BGNBND and the semantics of sending and processing generic bind options (see Section 4.4.3 on page 67)

Data Staging Area

Data Staging Area support is optional and will be described in a future Data Replication Reference.

1.1.4 What it Means to Implement DRDA Distributed Unit of Work

This section provides an overview of the functions and support that are required to implement DRDA Distributed Unit of Work distribution.

DRDA DUOW is made up of the following functions or support:

- DRDA Remote unit of work
- Distributed unit of work
- VAX and IEEE (non-byte reversed) ASCII machine types
- Multi-row Fetch
- Multi-row Insert
- Scrollable cursors
- Bind and Rebind options for I/O parallelism
- CCSID Manager

The first two functions listed are required functions for DRDA Distributed Unit of Work. While not being directly tied to the type of distribution being supported, the rest of the functions require SQLAM Level 4 and so are often also associated with DUOW. Of these other functions, only the VAX and IEEE (non-byte reversed) ASCII machine types are required. Although a function may be optional, it does require some amount of support in the DRDA components to allow these optional functions to exist in the DRDA Distributed Unit of Work environment.

Remote Unit of Work

Functionally, DRDA Remote Unit of Work is a proper subset of DRDA Distributed Unit of Work. To implement DRDA RUOW, implement only the DRDA RUOW functions. The functions that are DRDA DUOW are marked in the text below.

Distributed Unit of Work

Distributed unit of work consists of support for the following:

- On the application requester:
  - CMMRQSRM (see Section 4.4.12.2 on page 120)
  - RDBUPDRM (see Section 4.4.12.2 on page 120)
  - CMDVLTRM (see Section 4.4.12.2 on page 120, Table 11-1 on page 356, and Table 11-7 on page 365)
  - Two-phase commit protocols (see Section 3.1.4 on page 32, **Commit and Rollback Scenarios** on page 122, Section 12.7.3.4 on page 409, Section 12.7.6 on page 411, Section 12.7.7 on page 412, and Section 12.7.9 on page 413)
  - CRRTKN semantics and alert support (see Section 11.2.2.2 on page 353)
  - Coexistence rules (see Section 4.4.12.2 on page 120, Section 12.7.8 on page 413, Section 12.7.9 on page 413)
  - CMDVLT alert (see Table 11-1 on page 356 and Table 11-7 on page 365)
- On the application server:
  - CMMRQSRM (see Section 4.4.12.2 on page 120)
  - RDBUPDRM (see Section 4.4.12.2 on page 120)
  - CMDVLTRM (see Section 4.4.12.2 on page 120)
  - Two-phase commit protocols (see Section 3.1.4 on page 32, Section 4.4.12.2 on page 120, Section 12.7.3.4 on page 409, Section 12.7.6 on page 411, and Section 12.7.7 on page 412)
  - CRRTKN (semantics and alert support) (see Section 11.2.2.2 on page 353, Table 11-1 on page 356, and Table 11-7 on page 365)
  - CMDVLT alert (see Table 11-1 on page 356 and Table 11-7 on page 365)

VAX and IEEE ASCII (Non-Byte Reversed) Machine Types

The support for VAX and IEEE ASCII (non-byte reversed) machine types are required in DRDA and consist of the following:

- On the application requester:
  - Support for QTDSQLVAX (see Chapter 5 on page 137)
  - Support for QTDSQLASC (see Chapter 5 on page 137)
- On the application server:
  - Support for QTDSQLVAX (see Chapter 5 on page 137)
  - Support for QTDSQLASC (see Chapter 5 on page 137)

Multi-Row Fetch

The support for multi-row fetch is optional in DRDA. If supported, it consists of the following:

- On the application requester:
  - — Support for *nbrrow* parameter on CNTQRY (see Section 4.4.6.1 on page 78), which includes the ability to receive multiple rows when using fixed-row protocol.
  - — Sending FETCH on bind (see rule PB28 in Section 7.10 on page 296)
- On the application server:
  - — Receipt of *nbrrow* parameter on CNTQRY and the semantics of receiving it (see Section 4.4.6.1 on page 78)
  - — Receipt of FETCH on bind along with the semantics of receiving it (see rule PB28 in Section 7.10 on page 296)

If not supported, the application server must still support the rejection of FETCH at BIND time (see rule PB28, Section 7.10 on page 296).

Multi-Row Insert

The support for multi-row insert is optional in DRDA. If supported, it consists of the following:

- On the application requester:
  - — Support for *nbrrow* parameter on EXCSQLSTT (see Section 4.4.7 on page 96)
  - — Support for SQLDTAMRW multi-row insert RLO descriptor (see Section 5.2.3 on page 143)
- On the application server:
  - — Receipt of *nbrrow* parameter on EXCSQLSTT and the semantics of receiving it (see Section 4.4.7 on page 96)
  - — Support for SQLDTAMRW multi-row insert RLO descriptor (see Section 5.2.3 on page 143)

Scrollable Cursors

The support for scrollable cursors is optional in DRDA. If supported, it consists of the following:

- On the application requester:
  - — Support for *qryrelscr* parameter on CNTQRY (see Section 4.4.6.1 on page 78)
  - — Support for *qryrownbr* parameter on CNTQRY (see Section 4.4.6.1 on page 78)
  - — Support for *qryrfrtbl* parameter on CNTQRY (see Section 4.4.6.1 on page 78)
  - — Sending FETCH on bind (see rule PB28 in Section 7.10 on page 296)
- On the application server:
  - — Receipt of *qryrelscr* parameter on CNTQRY and the semantics of receiving it (see Section 4.4.6.1 on page 78)
  - — Receipt of *qryrownbr* parameter on CNTQRY and the semantics of receiving it (see Section 4.4.6.1 on page 78)
  - — Receipt of *qryrfrtbl* parameter on CNTQRY and the semantics of receiving it (see Section 4.4.6.1 on page 78)

  - — Receipt of FETCH on bind and the semantics of receiving it (see rule PB28 in Section 7.10 on page 296)

If not supported, the application server must still support:

- Rejection of FETCH at BIND time (see rule PB28 in Section 7.10 on page 296)

Bind and Rebind Options for I/O Parallelism

The support for bind and rebind options for I/O parallelism is optional in DRDA. If supported, it consists of the following:

- On the application requester:
  - — Support for *dgrioprl* parameter on BGNBND (see Section 4.4.3 on page 67)
  - — Support for *dgrioprl* parameter on REBIND (see Section 4.4.5 on page 74)
- On the application server:
  - — Support for *dgrioprl* parameter on BGNBND (see Section 4.4.3 on page 67)
  - — Support for *dgrioprl* parameter on REBIND (see Section 4.4.5 on page 74)

If not supported, the application server must still support:

- Receipt of *dgrioprl* on BGNBND and REBIND.

CCSID Manager

The support for CCSID manager is optional in DRDA. If supported, it consists of the following:

- On the application requester:
  - — Support for specifying CCSIDMGR on EXCSAT (see Section 4.3.1.13 on page 46 and Section 4.4.1 on page 54)
  - — Support for DDM character command parameters in CCSIDs 819, 850, and 500. It might also support other CCSIDs. (See Section 4.3.1.13 on page 46 and Section 4.4.1 on page 54.)
- On the application server:
  - — Support for specifying CCSIDMGR on EXCSATRD (see Section 4.3.1.13 on page 46 and Section 4.4.1 on page 54)
  - — Support for DDM character command parameters in CCSIDs 819, 850, and 500. It might also support other CCSIDs. (See Section 4.3.1.13 on page 46 and Section 4.4.1 on page 54.)

1.2 The FD:OCA Reference

The FD:OCA Reference describes the functions and services that make up the Formatted Data Object Content Architecture (FD:OCA). This architecture makes it possible to bridge the connectivity gap between environments with different data types and data representation methods by providing constructs that describe the data being exchanged between systems.

The FD:OCA is embedded in the Distributed Relational Database Architecture, which identifies and brackets the Formatted Data Object in its syntax. DRDA describes the connectivity between relational database managers that enables applications programs to access distributed relational data and uses FD:OCA to describe the data being sent to the server and/or returned to the requester. For example, when data is being sent to the server for inserting into the database or being returned to the requester as a result of a database query, the data type (character, integer, floating point, and so on) and its characteristics (length, precision, byte-reversed or not, and so on) are all described by FD:OCA.

The FD:OCA Reference is presented in three parts:

- Overview material to give the reader a feel for FD:OCA. This material can be skimmed.
- Example material that shows how the FD:OCA mechanisms are used. This should be read for understanding.
- References to the detailed FD:OCA descriptions. A few of these topics should be read up front to gain experience with the style of presentation and the content of the first several triplets. The rest can be read when the level of detail presented in that chapter is required. This is reference material.

1.3 DDM Reference

The DDM Reference describes the architected commands, parameters, objects, and messages of the DDM data stream. This data stream accomplishes the data interchange between the various pieces of the DDM model.

DDM Guide

Although there are many concepts and terms in Distributed Data Management, there are only a few that are key to the task of implementing a DRDA product. The suggested reading order for the DDM material should provide a good starting point in understanding the DDM terms used in DRDA. The intent is not to provide a complete list of DDM terms used by DRDA. For more detailed information, see the DDM Reference.

Key DDM Concepts

The first task in dealing with Distributed Data Management (DDM) is to obtain some background information to help place DDM in context. Table 1-1 lists the description and modeling terms that provide the necessary background information in understanding DDM.

Table 1-1 DDM Modeling and Description Terms

DDM Term	Term Title
DDM	Distributed Data Management Architecture
CONCEPTS	Concepts of DDM Architecture
OOPOVR	Object-oriented programming overview
INHERITANCE	Class inheritance
SUBSETS	Architecture subsets
EXTENSIONS	Product extensions to DDM Architecture
LVLCMP	Level compatibility

Key DDM Concepts for DRDA Implementation

After becoming familiar with the DDM overview terms in Table 1-1 on page 13, it needs to be understood that every DRDA implementation needs to provide the DDM components and model structures listed in Table 1-2. The concept of a component is described in the overview term for that component.

Table 1-2 DDM Terms of Interest to DRDA Implementers

DDM Term	Term Title
SQL	Structured Query Language
RDBOVR	Relational database overview
RDB	Relational database
SQLAM	SQL Application Manager
FDOCA	Formatted Data Object Content Architecture (FD:OCA)
SQLDTA	SQL program variable data
MGROVR	Manager layer overview
CMNOVR	Communications overview
CMNLYR	Communications layers
CMNMGR	DDM communications manager
CMNAPPC	LU 6.2 conversational communications manager (introduced in DRDA Level 2)
CMNSYNCPT	LU 6.2 sync point conversational communications manager (introduced in DRDA Level 2)
SYNCPTMGR	Sync point manager
AGENT	Agent
DSS	Data Stream Structures
OBJOVR	Object layer overview
SUPERVISOR	Supervisor
DICTIONARY	Dictionary
SECMGR	Security manager
DCESECOVR	DCE security overview

DDM Command Objects in DRDA

Another important aspect in implementing DRDA is to understand the DDM command objects used to flow the DRDA. The command objects are part of the DDM Relational Database (RDB) model. Table 1-3 lists these command objects and groups them by function. None of the parameters or parameter values associated with each command object are shown.

Table 1-3 DDM Command Objects Used by DRDA

DDM Term	Term Title
Connection establishment to a remote database manager	
EXCSAT	Exchange server attributes
ACCRDB	Access RDB
Package creation/rebind/remove	
BGNBND	Begin binding of a package to an RDB
BNDSQLSTT	Bind SQL Statement to an RDB package
ENDBND	End binding of a package to an RDB
REBIND	Rebind an existing RDB package
DRPPKG	DROP a package at an RDB
Query Processing	
OPNQRY	Open query
CNTQRY	Continue query
CLSQRY	Close query
Prepare/describe/execute SQL statements	
PRPSQLSTT	Prepare SQL statement
DSCSQLSTT	Describe SQL statement
DSCRDBTBL	Describe RDB table
DSCPVL	Describe User privileges
EXCSQLSTT	Execute SQL statement
EXCSQLIMM	Execute immediate SQL statement
Commit/rollback unit of work	
RDBCMM	RDB commit unit of work used by RUOW connections
RDBRLLBCK	RDB rollback unit of work used by RUOW connections
SYNCCTL	Sync point control request used for DUOW connections
SYNCRSY	Sync point resynchronization request used by DUOW connections
Security processing	
ACCSEC	Access security
SECCHK	Security check
Propagating special register settings	
EXCSQLSET	SET SQL environment
Connection establishment to a remote database manager	
EXCSAT	Exchange server attributes
ACCRDB	Access RDB

Reply Objects and Messages

Table 1-4 gives a list of the normal DDM reply data objects. These include reply messages, reply data, override data, query data descriptors, and query answer set data.

Table 1-4 DDM Reply Data Objects Used by DRDA

DDM Term	Term Title
EXCSATRD	Server attributes reply data
ACCRDBRM	Access to RDB completed
OPNQRYRM	Open query complete
ENDQRYRM	End of query condition
ENDUOWRM	End unit of work condition
RDBUPDRM	Update at an RDB condition (Introduced in DRDA Level 2)
SQLCARD	SQL communications area reply data
SQLDTARD	SQL data reply data
TYPDEFNAM	Data type definition name
TYPDEFOVR	Data type definition override
RSLSETRM	RDB result set reply message
SQLRSLRD	SQL result set reply data
QRYDSC	Query answer set description
QRYDTA	Query answer set data
ACCSECRD	Access security reply data
SECCHKRM	Security check complete reply message
SECTKN	Security token reply data
SYNCLOG	Identifies the sync point log used for a unit of work
SYNCCRD	Sync point control reply data in support of distributed unit of work
SYNCRRD	Sync point resynchronization reply data in support of distributed unit of work
SQLCINRD	SQL Result Set Column Info Reply Data
SQLDARD	SQLDA Reply Data

Open Group Technical Standard

Part 1:
Database Access Protocol

The Open Group

Chapter 2

Introduction to DRDA

Distributed Relational Database Architecture (DRDA) is the architecture that meets the needs of application programs requiring access to distributed *relational data*. This access requires *connectivity* to and among *relational database* managers operating in like or unlike operating environments. Structured Query Language (SQL) is the language that application programs use to access distributed relational data. DRDA is the architecture that provides the needed connectivity.

2.1 DRDA Structure and Other Architectures

DRDA requires the following architectures:

- *Distributed Data Management (DDM) Architecture*
- *Formatted Data Object Content Architecture (FD:OCA)*

DRDA uses *Character Data Representation Architecture (CDRA).* DRDA describes its use of *Logical Unit type 6.2 (LU 6.2)* and *Transmission Control Protocol/Internet Protocol (TCP/IP)* for network support, *SNA Management Services Architecture (MSA)* for problem determination support, and The Open Group *Distributed Computing Environment (DCE)* security support.

For a better understanding of DRDA, some familiarity with these architectures is useful. See **Referenced Documents** on page xxiv for a list of references that can provide helpful background reading about these architectures.

DRDA uses DDM, FD:OCA, and CDRA as architectural building blocks. DRDA also assumes the use of a network protocol and network management protocol as pieces of the architectural building blocks. The specific form of each of the blocks is specified to ensure that system programmers implement them in the same way for the same situations so that all programmers can understand the exchanges. DRDA ties these pieces together into a data stream protocol that supports this distributed cooperation.

2.2 DRDA and SQL

SQL is the database management system language and provides the necessary consistency to enable distributed data processing across like or unlike operating environments. It allows users to define, retrieve, and manipulate data across unlike environments. SQL provides access to distributed relational data among interconnected systems that can be at different locations.

DRDA supports SQL as the standardized *Application Programming Interface (API)* for execution of applications and defines flows (logical connections between the application and a database management system) that the program preparation process can use to bind SQL statements for a target *relational database management system (DBMS).*

An application uses SQL to access a relational database. When the requested data is remote, the function receiving the application SQL request must determine where the data resides and establish connectivity with the remote relational database system. One method used to make this determination is the SQL CONNECT statement. An application using the CONNECT statement directs the function receiving the application request to establish connectivity with a named relational database system. The term that DRDA uses to represent the name of the *relational database (RDB)* is RDB_NAME. The definition of RDB_NAME can be found in Section 6.2 on page 271.

Note:

> A relational database system can have multiple RDB_NAMEs, where each RDB_NAME represents a subset of the data managed by the relational database system.

Also, SQL includes RDB_NAME as the high order qualifier of relational database objects managed by the relational database system. See Section 6.3 on page 271 for details.

2.3 DRDA Connection Architecture

Connectivity in support of remote database management system processing requires a connection architecture that defines specific flows and interactions that convey the intent and results of remote database management system processing requests. DRDA provides the necessary connection between an application and a relational database management system in a distributed environment.

DRDA uses other architectures to describe what information flows between participants in a distributed relational database environment.[1] It also describes the responsibilities of these participants and specifies when the flows should occur. DRDA provides the formats and *protocols* required for distributed database management system processing, but does not provide the Application Programming Interface (API) for distributed database management system processing.

2.4 Types of Distribution

There are three degrees of distribution of database management system functions. Each degree of distribution has different DRDA requirements. Figure 2-1 on page 21 illustrates the degrees of distribution.

1. The terms *distributed database* and *distributed relational database* have the same meaning in this reference and are used interchangeably. The term *database* always means relational database.

Application-Directed Remote Unit of Work (DRDA Level 1)

— 1 DBMS per unit of work

— Multiple requests per unit of work

— 1 DBMS per request

— Application initiates commit

— Commitment at a single DBMS

Application-Directed Distributed Unit of Work (DRDA Level 2)

— Several DBMSs per unit of work

— Application directs the distribution of work

— Multiple requests per unit of work

— 1 DBMS per request

— Application initiates commit

— Commitment coordination across multiple DBMSs

Database-Directed Access (DRDA Level 4)

— Several DBMSs per unit of work

— Application directs requests to a DBMS

— DBMS distributes the unit of work to multiple DBMSs

— Multiple requests per unit of work

— 1 DBMS per request

— Application initiates commit

— Commitment coordination across multiple DBMSs

— Propagate special registers

Figure 2-1 Degrees of Distribution of Database Function

The degrees of distribution are:

- *Application-Directed Remote Unit of Work*

  With *Remote Unit of Work*, an application program executing in one system can access data at a remote database management system using the SQL supported by that remote database management system. Remote Unit of Work supports access to one database management system within a unit of work.[2] The application can perform multiple SQL statements within the unit of work. When the application is ready to commit the work, it initiates the commit at the database management system that is accessed for the unit of work. In the next unit of work, the application can access the same database management system or another database management system.

- *Application-Directed Distributed Unit of Work*

  With *Distributed Unit of Work*, within one unit of work, an application executing in one system can direct SQL requests to multiple remote database management systems using the SQL supported by those systems. However, all objects of a single SQL statement are constrained to be at a single database management system.

  When the application is ready to commit the work, it initiates the commit, and commitment coordination is provided by a *synchronization point manager*.

  Distributed Unit of Work allows:

  — Update access to multiple database management systems in one unit of work

  — Update access to a single database management system with read access to multiple database management systems, in one unit of work

  Whether an application can update multiple database management systems in a unit of work is dependent on the existence of a synchronization point manager at the application's location, synchronization point managers at the remote systems, and two-phase commit protocol support between the application's location and the remote systems. Two-phase commit protocols are discussed later.

- *Database-Directed Distributed Unit of Work*

  In database-directed requests, an application connects to a relational database management system (RDB) that can execute one or more SQL requests locally or route some or all of the SQL requests to other RDBs. The RDB determines which system manages the data referenced by the SQL statement and automatically propagates any special registers set by the application and directs the request to that system. The DBMS is expected to support DUOW connections, but allows restricted connectivity if it supports RUOW connections.

2.5 DRDA Protocols and Functions

At the Remote Unit of Work level, DRDA supports the connection between an application process and the application server of a database management system (DBMS).

At the Distributed Unit of Work level, DRDA supports the connection between an application process to application servers of multiple DBMSs, as well as an application server to multiple database servers of multiple DBMSs.

DRDA provides one kind of connection protocol and two basic kinds of functions.

The connection protocol is:

- *Application Support Protocol.* Provides connection between application requesters (AR) and application servers (AS).

  The application requester supports the application end of the DRDA connection by making requests to the application server, while the application server supports the database management system end by answering these requests.

2. A unit of work can also be known as a *transaction*.

- *Database Support Protocol*. Provides connections between application servers and database servers (DS). Prior to executing any SQL statements at a database server, special register settings set by the application must be propagated to the database server.

The function types are:

- *Application Requester Functions*. Support SQL and program preparation services from applications.
- *Application Server Functions.* Support requests that application requesters (ARs) have sent and routes requests to database servers by connecting as an application requester.
- *Database Server Functions*. Support requests from application servers. Support the propagation of special register settings.

These three functions are illustrated in Figure 2-2.

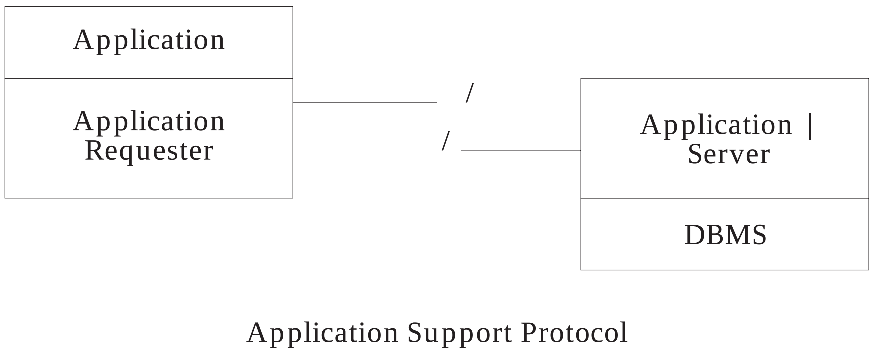


Figure 2-2 DRDA Network

A single system can implement all of the functions. Such a system would behave appropriately (differently) according to the role it is playing for any particular request.

This relationship is illustrated in Figure 2-3 on page 24.

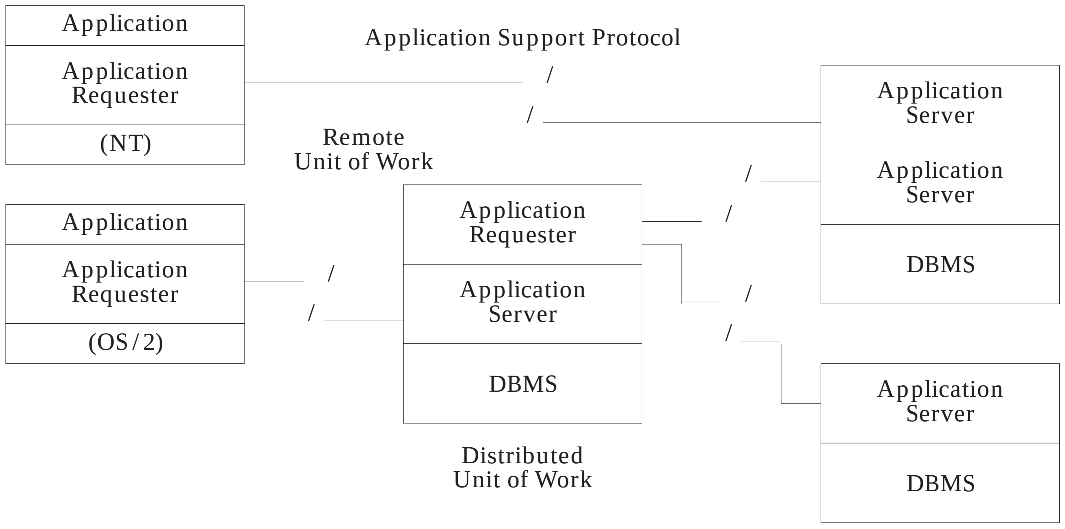


Figure 2-3 DRDA Network Implementation Example

Any database management systems could be in any position in this figure. Figure 2-3 shows three places where databases (DBMS) from different vendors—for example, IBM, Microsoft, Oracle, and so on—may be used. Implementations are based upon business requirements.

A developer might choose to implement either DRDA Remote Unit of Work or Distributed Unit of Work. If a Remote Unit of Work component is being developed, all functions should be implemented except those identified as DRDA Distributed Unit of Work. An implementer of a subset of these functions is not required to support Distributed Unit of Work.

Some functions of this architecture are optional. These are defined in Section 1.1 on page 2, as well as the optionality of the DDM commands, replies, and parameters as defined in the DDM Reference.

This volume describes both Remote Unit or Work and Distributed Unit of Work, plus additional functions that have been included since DRDA was first introduced. The type of support used is dependent on the DRDA manager levels in use. See Chapter 4 on page 37 for details on DDM managers.

Chapter 3

Using DRDA—Overall Flows

DRDA flows are high-level communication paths that pass information between application environments and database management systems. Because these flows cross underlying architecture boundaries, this chapter relates individual pieces of flows to the defining architecture.

The rest of this volume describes the details of how DRDA uses each of the underlying architectures and discusses each of these flows in greater detail.

3.1 Introduction to Protocol Flows

Application support protocol flows establish and define the connections for the information exchange from executing application programs and application development programs to database management systems.

The logical flow figures in this chapter are examples of the type of information that flows between an application requester and an application server in support of a DRDA activity. The figures refer to information flowing from the application requester as application end information and to information flowing from the application server as database management system end information. The arrows depict the direction of flow as opposed to the actual time of flow.

Each figure uses verbs, commands, and terms from the underlying architectures. For the sake of example, we assume the use of SNA as the network protocol in the example flows of this chapter. Refer to Chapter 12 on page 385 for an explanation of the use of SNA verbs. Refer to Chapter 4 on page 37 for an explanation of the use of DDM commands and terms. Refer to Chapter 5 on page 137 for an explanation of the use of FD:OCA constructs.

3.1.1 Initialization Flows

An initialization flow occurs before, or as part of the response to, the first remote request from an application program or an end user. The first remote request can be an explicit SQL CONNECT statement, or an implicit SQL CONNECT due to some other SQL statement that implies a connect to the database management system. Using DRDA Remote Unit of Work, an application or end user can only connect to one relational database per unit of work. Using Distributed Unit of Work, an application or end user can connect to multiple relational databases in a single unit of work, but only one SQL connection is current at any time. The application or end user defines which SQL connection is current through SQL calls. The target database of the SQL CONNECT statement can be a local database, in which case DRDA protocols are not in use. An initialization flow creates a network connection and prepares a remote DRDA execution environment for executing a DRDA request.

A successful initialization flow results in an authenticated network connection between specific products at understood release levels. Authentication processing is required in DRDA. Authentication can occur by one or more of the following techniques:

- Through Distributed Computing Environment (DCE) security mechanisms
- Passing a userid and password in a DDM command

- Passing a userid only (equivalent to already verified) in a DDM command
- Passing a userid, password, and new password in a DDM command
- During SNA initialization processing through use of LU-LU Verification (Partner-LU verification) and Conversation Level Security (End-user verification) as specified in the SNA architecture

An initialization flow also propagates information for accounting and problem determination. For example, the initialization flow specifies an end-user name and a unique correlation token. In addition, TCP/IP connections provide a unique DDM unit of work identifier (UOWID), the server's IP address and PORT number. SNA connections provide a unique SNA logical unit of work identifier (LUWID), server's LU name, and transaction program name (TPN). These provide the who, what, when, and where information useful for accounting in DRDA environments.

A DRDA initialization flow uses a network protocol and DDM commands to create a connection to a relational database.

Figure 3-1 on page 27 and Figure 3-2 on page 28 show the logical flow of information between an application requester and an application server. Arrows depict the direction that information flows rather than the time that actual physical flows occur on the link.

Figure 3-1 on page 27 assumes SNA security is used. Figure 3-2 on page 28 assumes the security information is carried in DDM commands and responses. For this example, it is assumed that DCE security is in use. Other possible security information, which along with DCE security is likely for TCP/IP networks are:

- Userid and password
- Userid only
- Userid, password, and new password

Both figures assume:

- An SQL CONNECT statement was the remote request that started the initialization flow.
- The application server supports the DDM TYPDEF that the application requester specified. The DDM TYPDEF specifies the data representation used when transmitting parameters and values between the application requester and the application server. Other remote requests (such as DDM BGNBND) cause similar flows of information.
- All SNA verbs and DDM commands execute successfully

```
Application End Information              DBMS End Information

I am USER_ID;                      --->
I desire to connect to DRDA TPN
at NETID.LU_NAME using MODE_NAME;
I am part of LUWID
(SNA ALLOCATE)

I am RELEASE of AR from DRDA
ENVIRONMENT and request the
following DDM managers
(DDM EXCSAT)
                                   <---   I am RELEASE of AS from DRDA
                                          ENVIRONMENT and support
                                          the following subset of
                                          requested DDM managers
                                          (DDM EXCSATRD)
I wish to access RDB_NAME          --->
using DRDA flows with TYPDEF
(DDM ACCRDB)
                                   <---   I support the TYPDEF specified
                                          I will use DRDA flows with
                                          TYPDEF
                                          (DDM ACCRDBRM)
```

Figure 3-1 Logical Flow: Initialization Flows with SNA Security

```
Application End Information              DBMS End Information

I desire to connect to DRDA TPN   --->
at NETID.LU_NAME using MODE_NAME;
I am part of LUWID
(SNA ALLOCATE)

I am RELEASE of AR from DRDA
ENVIRONMENT and request the
following DDM managers
(DDM EXCSAT)
                                  <---   I am RELEASE of AS from DRDA
                                         ENVIRONMENT and support
                                         the following subset of
                                         requested DDM managers
                                         (DDM EXCSATRD)
Here is the security mechanism    --->
I wish to use
(DDM ACCSEC)
                                  <---   I accept your security
                                         mechanism
                                         (DDM ACCSECRD)
I am USER_ID and here is          --->
information to authenticate me
(DDM SECCHK)
                                  <---   I accept who you are
                                         and here is information to
                                         authenticate me
                                         (DDM SECCHKRM)
I accept who you are and          --->
I wish to access RDB_NAME
using DRDA flows with TYPDEF
(DDM ACCRDB)
                                  <---   I support the TYPDEF specified
                                         I will use DRDA flows with
                                         TYPDEF
                                         (DDM ACCRDBRM)
```

Figure 3-2 Logical Flow: Initialization Flows with DCE Security

In Figure 3-1 on page 27 and Figure 3-2 on page 28, the material in parentheses shows the SNA verbs and DDM commands and responses that carry the information.

For a more in-depth description of the initialization processing flows, see:

- For SNA, Figure 12-1 on page 396 and Figure 12-3 on page 398
- For TCP/IP, Figure 13-2 on page 425

3.1.2 Bind Flows

A DRDA bind flow results in the creation and storage of a package at an application server.

DRDA bind flows use a network protocol, DDM, FD:OCA, and CDRA. Figure 3-3 on page 30 shows the type of information that flows between an application requester and an application server. Arrows depict the direction that information flows rather than the time that physical link flows occur. Figure 3-3 on page 30 assumes:

- The connection has been established.
- The application requester is binding two SQL statements with application variable definitions into a single package at the application server.

```
Application End Information              DBMS End Information

I desire to Bind SQL                ---> 
statements to PACKAGE with
CONSISTENCY TOKEN using the
following Bind options and
Parser options.
(DDM BGNBND)
                                    <---   I executed BGNBND with the
                                           following results
                                           (DDM SQLCARD using FD:OCA)
Bind SQL STATEMENT as SECTION       --->
in PACKAGE with CONSISTENCY
TOKEN referencing the following
application program host language
variable declarations
(DDM BNDSQLSTT, DDM SQLSTT,
and DDM SQLSTTVRB using FD:OCA)
                                    <---   I executed BNDSQLSTT with the
                                           following results
                                           (DDM SQLCARD using FD:OCA)
Bind SQL STATEMENT as SECTION       --->
in PACKAGE with CONSISTENCY
TOKEN referencing the following
application program host language
variable declarations
(DDM BNDSQLSTT, DDM SQLSTT,
and DDM SQLSTTVRB using FD:OCA)
                                    <---   I executed BNDSQLSTT with the
                                           following results
                                           (DDM SQLCARD using FD:OCA)
I have completed BIND               --->
(DDM ENDBND)
                                    <---   I executed ENDBND with the
                                           following results
                                           (DDM SQLCARD using FD:OCA)
```

Figure 3-3 Logical Flow: Bind Flows

For a more in-depth description of the bind flows, see:

- For SNA, Figure 12-6 on page 403
- For TCP/IP, Figure 13-3 on page 427

3.1.3 SQL Statement Execution Flows

A DRDA SQL statement execution flow transmits a DDM command to an application server that requests a relational database management system to execute an SQL statement and returns the results to the application requester. There are several types of statement execution flows. Figure 3-3 on page 30 is an example of the flow that executes a previously bound SQL statement involving a cursor and uses the DDM commands OPNQRY, CNTQRY, and CLSQRY to perform functions analogous to the SQL cursor statements OPEN, FETCH, and CLOSE. If an application server determines the SQL statement is for another RDB, the application server must propagate any special registers set or changed by the application since the last request to that database server (DS).

DRDA remote SQL statement requests often operate on multiple rows of multiple tables and can cause the transmission of multiple rows from the application server to the application requester. DRDA provides two data transfer protocols in support of these operations:

- Fixed row protocol[3]
- Limited block protocol

The fixed row protocol guarantees the return of exactly the number of rows the application requested, or the number of rows available if it is less than the number of rows the application requested, whenever the application requester receives row data. The limited block protocol optimizes data transfer by guaranteeing the transfer of a minimum amount of data (that can be part of a row, multiple rows, or multiple rows and part of a row) in response to each DRDA request. Application requesters and application servers can use the limited block protocol for the processing of a query that uses a cursor for read-only access to data.

See the terms FIXROWPRC[4] and LMTBLKPRC in the DDM Reference for more details on fixed row and limited block protocols.

DRDA SQL statement execution flows use a network protocol, DDM, FD:OCA, and CDRA.

Figure 3-4 on page 32 shows the type of information that flows between an application requester and an application server. Arrows depict the direction that information flows rather than the time that physical link flows occur.

This figure assumes that:

- The connection has been established.
- An OPEN and FETCH SQL statement sequence was the remote request that caused the SQL statement execution flow to occur.

3. In DRDA Level 1 this was known as *single row protocol*. DRDA Level 2 introduced the optional support for multi-row fetches. Single row fetch (single row protocol) is the default and is also a special case of fixed row protocol. DRDA application requesters and application servers supporting only Remote Unit of Work are not required to support multi-row fetches.

4. The default for fixed row protocol is known as *single row protocol* (or *single row fetch*), and can be specified using the term SNGROWPRC.

- The query does not require application input variable values.
- The query processing uses the limited block protocol and that query processing requires the transmission of two blocks containing row data.

```
Application End Information              DBMS End Information

I desire to Open Query for        --->
SECTION of PACKAGE with
CONSISTENCY TOKEN using
BLOCKSIZE
(DDM OPNQRY)
                                  <---   I executed the OPNQRY using
                                         QUERY PROTOCOl TYPE with
                                         the following results
                                         (DDM OPNQRYRM with DDM QRYDSC
                                         and DDM QRYDTA using FD:OCA)
Continue query processing         --->
for SECTION of PACKAGE with
CONSISTENCY TOKEN using
BLOCKSIZE
(DDM CNTQRY)
                                  <---   I executed the CNTQRY with
                                         the following results
                                         (DDM QRYDTA using FD:OCA
                                         and DDM ENDQRYRM
                                         with DDM SQLCARD using FD:OCA)
```

Figure 3-4 Logical Flow: SQL Statement Execution Flows

For a more in-depth description of the actual DRDA execute SQL statement flows, see:

- For SNA, Figure 12-9 on page 406
- For TCP/IP, Figure 13-4 on page 428

3.1.4 Commit Flows

A successful commit of the application's work involves a coordinated commitment of all work processed by the application since the last successful commit or startup of the application. This process is also known as *resource recovery processing*, and the point where all resources are in a consistent state is called a *synchronization point*. The flows involved with the commitment of resources are dependent on the sync point manager in use between the application end and DBMS end of the connection. The SNA communication sync point manager supports protected network connections which use SNA-defined two-phase commit flows to commit the work. The DRDA sync point manager uses DDM-defined sync point control flows to commit the work. DDM flows are independent of the underlying communications manager.[5]

5. DDM sync point manager supports presumed abort and implied forget processing to optimize performance, eliminating all logging requirements at an application requester. Also, optional resync server flows are defined to eliminate all logging requirements for an unsecure requester. Refer to the SYNCPTOV term in the DDM Reference for an overview of DRDA's two-phase commit processing.

Remote Unit of Work connections use DRDA defined one-phase commit or two-phase commit flows to commit the work. The type of DRDA commit flow used is dependent on the level of the sync point manager identified during initialization. Without the support of a sync point manager, DRDA one-phase commit is used to coordinate all commits. For work that involves both protected and unprotected network connections, the application requester participates in the processing of both flows. For information about committing work on Distributed Unit of Work connections, refer to Figure 12-11 on page 409.

Figure 3-5 shows the type of information that flows between an application requester and application server to commit work using DRDA two-phase flows.

```
Application End Information              DBMS End Information

I want to start a new unit of     --->
work by sending the new
unit of work identifier
(DDM SYNCCTL new unit of
work identifier command)
                                  <---   Set identifier for current
                                         unit of work and participate
                                         in the next commit
Prepare for commitment of the     --->
current unit of work
(DDM SYNCCTL prepare to
commit command)
                                  <---   Prepare for commitment of the
                                         current unit of work
                                         (DDM SYNCCRD request to
                                         commit reply)
Commit the current unit of        --->
work
(DDM SYNCCTL committed
command)
                                  <---   Commit and forget the current
                                         unit of work
                                         (DDM SYNCCRD forget unit
                                         of work reply)
```

Figure 3-5 Logical Flow: DRDA Two-Phase Commit

Figure 3-6 on page 34 shows the information that flows between an application requester and application server to commit work using DRDA one-phase commit. In both flows, a request to commit the work can result in the application server roll backing the unit of work.

```
Application End Information           DBMS End Information

I want to commit the current      --->
unit of work
(DDM RDBCMM, DDM EXCSQLSTT, or
DDM EXCSQLIMM)
                                  <---   I committed the work
                                         (DDM ENDUOWRM with SQLCARD
                                         using FD:OCA)
```

Figure 3-6 Logical Flow: DRDA One-Phase Commit Using DDM Commands

The SQL application should explicitly use commit or rollback functions before termination. It is the responsibility of the application requester, however, to ensure commit and rollback functions are invoked at application termination. For details about committing work using SNA refer to Figure 12-13 on page 411, or for TCP/IP refer to Figure 13-7 on page 430.

3.1.5 Termination Flows

A successful termination flow results in the orderly close of the network connection between the application program or the end user and the DBMS. The termination of the network connection between an application requester and an application server terminates the application server.

The normal or abnormal termination of an application causes the application requester to initiate terminate network connection processing for the network connection associated with the execution of the application. The normal termination of a protected connection must be performed using a commit. The termination of an unprotected connection using DRDA two-phase commit flows must perform a commit that indicates the connection is to be released when the commit is successful. In terms of SNA, a DEALLOCATE on protected network connections must be followed by the SNA command SYNCPT before the conversation is deallocated. In terms of DRDA, a DDM sync point control is sent with the release connection indicator. The connection is not disconnected if the unit of work results in a rollback. The semantics of a disconnect of the network connection include an implied rollback. It is the responsibility of the application server to ensure a rollback occurs when a network failure is detected.

Figure 3-7 on page 35 shows the type of information that flows between an application requester and an application server to terminate a protected connection using DRDA two-phase flow.

```
Application End Information               DBMS End Information

Prepare for commitment of the     --->
current unit of work for a
released connection
(DDM SYNCCTL prepare to
commit with RLSCONV set
to TRUE)
                                  <---   Prepare for commitment of
                                         the current unit of work
                                         (DDM SFNCCRD request to
                                         commit reply)
Commit the current unit of        --->
work
(DDM SYNCCTL committed)
                                  <---   Commit and forget the current
                                         unit of work
                                         (DDM SYNCCRD forget unit of
                                         work reply)
Terminate the network             --->
connection
                                  <---   Terminate the network
                                         connection and application
                                         server process
```

Figure 3-7 Logical Flow: DRDA Two-Phase Commit Termination Flows Using DDM Commands

Figure 3-8 shows the type of information that flows between an application requester and an application server to deallocate a single SNA protected conversation.

```
Application End Information               DBMS End Information

I desire to disconnect from       --->
DRDA Transaction Program
(SNA DEALLOCATE, SNA SYNCPT)
                                  <---   Receive deallocate and commit
                                         notification. Commit unit of
                                         work and terminate application
                                         server process
```

Figure 3-8 Logical Flow: SNA Termination Flows on Protected Conversations

Chapter 4

The DRDA Processing Model and Command Flows

DRDA's set of models allows the separation of an application from the relational data it will process. If moving the data does not split it across systems, the process of moving the relational data from the system containing the application or the application from the system containing the relational data should not require changes to the application's source code to get the same results.

DRDA describes, through protocol models, the necessary interchanges between the application (or an agent on its behalf) and one or more remote relational databases[6] to perform the following functions:

- Establish a connection between an application and a remote relational database.
- Bind an application's host language variables and SQL statements to a remote relational database.
- Execute those bound SQL statements, on the behalf of the application, in the remote relational database and return the correct data or completion indication to the application.
- Execute dynamic SQL statements, on the behalf of an application, in a remote relational database and return the correct data or completion indication to the application.
- Maintain consistent unit of work boundaries between an application and one or more remote relational databases.
- Terminate the connection between an application and a remote relational database.

DRDA describes these functions as a series of commands and command replies that are sent between the application (or an agent on its behalf) and a remote relational database. DRDA also describes the correct flow of these commands and command replies between the application (or an agent on its behalf) and a remote relational database. Included in the description of the commands and flows are:

- Encoding/decoding rules for commands, parameters, and data
- Parameter values on commands and command replies:
  - Optional/required
  - Valid/not valid
  - Assigned (constant/defined values)
  - Defaults
- Valid command replies for each command
- Error messages valid for each command
- Recovery procedures for command error messages, when applicable
- Order in which commands can be sent

6. DRDA Remote Unit of Work is limited to one relational database per unit of work.

4.1 DDM and the Processing Model

The DDM model, DDM terms, and DDM architecture define the functions and the command flows that make up DRDA. To further explain the relationship of DRDA and DDM, this chapter:

- Presents the DDM processing model and relates it to SQL, relational database managers, and DRDA.
- Presents the DDM server model, including manager objects, and relates it to the DRDA model.
- Describes the normal flow of DDM commands (as examples) between an application requester and an application server to accomplish the tasks of:
  - — Establishing connectivity between the application requester and application server
  - — Determining the functional capabilities of the application server
  - — Binding SQL statements in an application to a remote relational database
  - — Dropping a set of bound SQL statements from a remote relational database
  - — Executing a bound Query against a remote relational database
  - — Executing SQL statements
  - — Completing/terminating a unit of work
  - — Terminating the connection between application requester and application server upon completion of the application
- Provides some examples of error conditions and the normal processing associated with them in an application requester or application server.
- References other sections of this document and other documents for descriptions of:
  - — DDM terms
  - — FD:OCA descriptors
  - — DRDA command usage rules
  - — SNA LU 6.2 two-phase commit protocols
  - — DDM two-phase commit protocols
  - — The DCE Security mechanisms

4.2 DRDA's Relationship to DDM

This chapter describes the use of the DDM architecture to perform DRDA functions. The details for DDM functions and examples are in the DDM references. The figures in this chapter contain the DDM commands used to perform each of the DRDA functions. The exact syntax and semantics of the DDM commands are described in the DDM Reference.

DDM is an architected data management interface used for data interchange among like or unlike systems. The DDM architecture is independent of an implementing system's hardware architecture and its operating system. DDM provides a conceptual framework or model for constructing common interfaces for data interchange between systems. The DDM data stream (which consists of architected commands, parameters, objects, and messages) accomplishes the data interchange between the various pieces of this model.

The references cited in **Referenced Documents** on page xxiv describe DDM in greater detail. The reader should be familiar with the DDM Reference before reading this chapter. The level of DDM documentation that should be referenced is dependent on the level of the DRDA implementation. For example, for DRDA Level 3, see the DDM Level 5 documentation.

DDM describes the model for distributed relational database processing between relational database management products. It also provides all the commands, parameters, data objects, and messages needed to describe the interfaces between the various pieces of that model.

DRDA describes the contents of all the data objects that flow on either commands or replies between the application requester (AR) and the application server (AS). The formats of these objects are described in Chapter 5 on page 137. The Formatted Data Object Content Architecture (FD:OCA) is used in that chapter as the underlying description architecture. FD:OCA is a powerful architecture for data description, and DRDA uses a subset of that architecture. In this volume, the terms *FD:OCA descriptor* (meaning a description expressed using FD:OCA) and the *unqualified word descriptor* are used interchangeably and mean the same thing. In either case, these terms refer to the DRDA data definitions included in Chapter 5 on page 137. FD:OCA data, in this volume, means data defined by DRDA descriptions.

DDM describes the common interfaces for the interchange of data between more data models than relational database management products support. Therefore, there are many more DDM commands, parameters, data objects, and reply messages (all of which are described as terms in the referenced documents) than are required in any implementation of distributed relational database management. This chapter describes which of the DDM terms are part of DRDA.

This chapter also describes additional restrictions on the use of some of the DDM terms in order to provide a consistent DRDA usage of DDM architecture, efficient implementations of DRDA, and a more understandable architecture for implementers of relational database management products. These restrictions are described in detail in Chapter 7 on page 281 and Chapter 5 on page 137. Additionally, the normal or usual usage of the DDM terms is described through the use of examples in Section 4.4 on page 54. For more background reading, see **DDM Guide** on page 13 for the suggested reading order of the DDM Reference.

4.3 The DRDA Processing Model

The system that contains an executing application that is requesting relational database management functions on another system is called the source system in the DDM processing model. The system that contains the relational database that provides the function is called target system in the DDM model. The DDM source system is referred to as the application requester in the DRDA processing model and the DDM target system is referred to as the application server in the DRDA processing model.

The DDM processing model is a set of managers that act on or organize data within a DDM data stream or within the manager itself. Figure 4-1 on page 47 shows all of the DDM managers whose functions are defined in DRDA. The DDM managers shown include all the entities in this illustration except the application, the relational database, and the communication support. The support that each of the managers provides is discussed in Section 4.3.1.

For further information on the DDM processing model, the DDM server model, and DDM server managers, see the introductory chapter of the DDM Reference. These sections also introduce some of the terms and terminology used in the DDM architecture.

4.3.1 DRDA Managers

The DDM processing model is composed of managers that are grouped together and function as servers. DRDA defines three DDM servers, the application requester (AR), the application server (AS), and the database server (DS).

The next sections discuss each DDM manager used in DRDA processing. There are descriptions for the function of the manager and the relationship between managers. Neither DDM nor DRDA defines the interfaces between DDM managers. Neither DDM nor DRDA requires an implementation to package the functions according to the DDM manager model.

A DRDA implementation of an application requester must be able to create the DDM commands, command parameters, and command data objects to be sent to an application server, and to receive the DDM reply messages and reply data objects that the application server returns. A DRDA implementation of an application server must be able to receive the commands, command parameters, and command data objects that an application requester has sent, and, based upon the command requests, generate the appropriate reply messages and reply data objects to be returned to the application requester.

The following sections discuss the managers that make up these servers.

4.3.1.1 SNA Communications Manager

The SNA LU 6.2 conversational communications manager (CMNAPPC) provides unprotected conversational support for the agent in an application requester or application server. It provides this support using conversational protocols that the local LU 6.2 communications facilities provide in accordance with the description provided in Chapter 12 on page 385.

This DDM communications manager is the program associated with the transaction program name (TPN) and an instance of it is created in the application server system when a request for an application server is received.

It manages the DRDA protocols and rules that are to be used on top of the LU 6.2 support.

It builds the DDM data streams from the commands, parameters, data, and replies that the agent passed to it. It also parses the DDM data stream (only to the data stream structure level) and passes commands, parameters, data, and replies to the agent.

4.3.1.2 SNA Sync Point Communications Manager

The SNA LU 6.2 sync point conversational communications manager (CMNSYNCPT) is introduced in DRDA Distributed Unit of Work and is not required to be supported for DRDA Remote Unit of Work. CMNSYNCPT provides protected conversational support for the agent in an application requester or application server. It provides this support using conversational protocols that the local SNA LU 6.2 communications facilities provide in accord with the description provided in Chapter 12 on page 385.

This is the program associated with the transaction program name (TPN), and an instance of it is created in the application server system when a request for an application server is received.

It manages the DRDA protocols and rules that are to be used on top of the LU 6.2 support.

It builds the DDM data streams from the commands, parameters, data, and replies that the agent passed to it. It also parses the DDM data stream (only to the data stream structure level) and passes commands, parameters, data, and replies to the agent.

4.3.1.3 TCP/IP Communications Manager

The TCP/IP communications manager (CMNTCPIP) provides TCP/IP network protocol support for the agent in an application requester or application server. It provides this support using TCP/IP protocols that the local TCP/IP communications facilities provide in accordance with the description provided in Chapter 13 on page 421.

This DDM communications manager is the program associated with the DRDA *well known port* and an instance of it is created in the application server system when a request for an application server is received.

It manages the DRDA protocols and rules that are to be used on top of the TCP/IP support.

It builds the DDM data streams from the commands, parameters, data, and replies that the agent passed to it. It also parses the DDM data stream (only to the data stream structure level) and passes commands, parameters, data, and replies to the agent.

4.3.1.4 Agent

The function of an agent (AGENT) is to represent a requester to a server. There is an instance of the agent present in both an application requester and an application or database server, and, although there are some common functions in the two agents, there are additional functions that depend on which of the agent environments is being considered.

In an application requester, the agent interfaces with the SQLAM to receive requests and pass back responses.

In an application or database server, the agent interfaces to managers in its local server to determine where the command should be sent to be processed, to allocate resources, to locate resources, and to enforce security.

The agent in the application or database server represents the requester to the local server's supervisor to control and account for the memory, processor, file-storage, spooling, and other resources a single user job/task uses.

The agent in an application or database server also represents the requester to its server's security manager. The application or database server's security manager validates each request the requester makes for a resource.

The agents in an application requester and in an application or database server interface to the DDM communications manager for any required communications.

To access remote relational databases, an agent in the application requester requests that the DDM communications manager establish communications with an agent in the application or database server on the system that owns the relational database. The agent in the application requester directs all following requests for that relational database to the agent in the application or database server.

4.3.1.5 Supervisor

The supervisor (SUPERVISOR) manages a collection of managers within a particular operating environment.

The supervisor provides an interface to its local system services such as resource management, directory, dictionary, and security services. The supervisor interfaces with the local system services and the other managers.

The only command defined for the supervisor is the Exchange Server Attributes (EXCSAT) command that allows two servers to determine their respective server class names and levels of support.

4.3.1.6 Security Manager

The security manager (SECMGR) is part of the DDM model to represent security functions. Not all security functions are defined in DRDA. Some of the details of security in DRDA are described in Chapter 10 on page 343.

The primary functions of a security manager include:

- Participation in end-user identification and authentication processing for the security mechanisms listed in Table 4-3 on page 62. (For example, DCE security, userid only, userid and password, and so on.)

  Note: If none of these security mechanisms are in use, the communications facilities must perform the end-user identification and authentication functions.

- Ensure that the requester, which the agent represents, is only allowed to access relational databases, commands, dictionaries, or directories in the manner for which it has been authorized.

  The authorization of a user to objects within a relational database is the responsibility of the relational database manager. DDM provides various reply messages for rejecting commands due to authorization failures.

4.3.1.7 Directory

A directory is an object that maps the names of instances of manager objects to their locations. The directory manager (DIRECTORY) provides support for locating the managers that make up a server (the AS).

The product, not DDM or DRDA, defines the interfaces to the directory manager.

4.3.1.8 Dictionary

A dictionary is a set of named descriptions of objects. A dictionary manager (DICTIONARY) provides interfaces to use the object descriptions that are stored in the dictionary.

In an application requester, the agent and SQLAM use the dictionary to create valid DDM command and data objects. They also use the dictionary to parse the reply data and messages that are returned from the application server to determine what manager should process them and what data is returned to the application.

In an application server, the agent and SQLAM use the dictionary to parse the DDM command and data objects that the application server has received from the application requester to determine which manager is to process the request and what type of processing is to be done. They also use the dictionary to construct valid reply messages and reply data to be returned to the application requester.

4.3.1.9 Resynchronization Manager

The resynchronization manager is the system component that recovers protected resources when a commit operation fails. If a commit operation fails and the outcome of the unit of work may be in doubt, the resynchronization manager initiates resynchronization flows to resolve in-doubt units of work.

Also, in conjunction with the sync point manager, the resynchronization manager provides resync server support. If supported, an unsecure requester can migrate resynchronization responsibilities to a server. The resync server logs unit of work state information and performs resynchronization on behalf of the requester.

4.3.1.10 Sync Point Manager

The sync point manager is the system component that coordinates commit and rollback operations among the various protected resources. With distributed updates, sync point managers on different systems cooperate to ensure that resources reach a consistent state. The protocols and flows used by sync point managers can also be referred to as two-phase commit protocols.

The sync point manager can be called directly by the application to commit the unit of work or by the SQLAM at the application requester on behalf of the application.

Sync point operations flow as DDM commands, objects, and replies using either TCP/IP or SNA communications manager. The agent forwards sync point commands and objects to the SYNCPTMGR which then interfaces to the RDB or SQLAM to perform sync point operations. Sync point replies are sent from the SYNCPTMGR to the agent in the form of a DDM reply and objects which are sent using the underlying communications manager.

For SNA protected conversations, the reference *SNA LU 6.2 Reference: Peer Protocols* (SC31-6808, IBM) describes in greater detail the functions of a sync point manager as well as its relationship to the resource managers and applications.

4.3.1.11 SQL Application Manager

The function of the SQL application manager (SQLAM) is to represent the application to the remote relational database manager. There is an instance of the SQLAM present in both an application requester and an application or an application server to a database server. An SQLAM performs functions depending on which environment the application is in.

The SQLAM handles all DRDA flows. This manager is responsible for ensuring that the application requester, application server, and database server are using the proper commands and protocols.

In the application requester, the SQLAM processes the requests it receives from the application or application server and invokes the corresponding function/operation from the SQLAM in the application server through DRDA commands. Neither DDM nor DRDA defines the interface that the SQLAM provides in the application requester for requests from the application.

In the application server, the SQLAM processes the requests it receives and invokes the corresponding function/operation from the relational database manager. It uses interfaces that the relational database manager and other managers in its environment have defined. The target SQLAM responds to the source SQLAM through architected reply messages and reply data.

Both source and target SQLAMs are responsible for data representation conversions as necessary for DDM command and reply message parameters. Unless overridden by the DDM Coded Character Set Identifier (CCSID) manager (CCSIDMGR), the character parameter values are represented in CCSID 500.

The SQLAM is the only manager in either the application requester, application server, or database server that understands the format of the command data objects and reply data objects. It is, therefore, responsible for creating and interpreting all of the FD:OCA descriptors that the application requester and application server pass between them.

The SQLAM at the application requester is responsible for converting numeric and character data, if necessary, before passing data values to the application programs.

The SQLAM at the application server or database server is responsible for converting numeric data, if necessary, before passing the data values to the relational database. The relational database is responsible for performing any necessary character conversion.

The SQLAM at the application server or database server registers with the sync point manager so that the SQLAM is kept informed of the status of the unit of work from the viewpoint of the global environment.[7]

The SQLAM at the application requester can call the sync point manager to begin the resource recovery process on behalf of the application. The application requester must also register with the sync point manager to allow the application requester to manage commit and rollback to the application servers that are not under sync point management control.

A list of the DDM commands that the SQLAM understands and handles, for DRDA flows, are described briefly in Table 4-1 on page 45. This table contains only those DDM commands that an implementation of DRDA flows requires. These commands are documented in the DDM Reference as commands (and corresponding code points) of the same names.

DDM Guide on page 13 contains a full list of the DDM terms that are required to implement the DRDA flows. Optional commands are indicated in the OPT'L column. See Section 7.9 on page 295 for details on optional commands. Section 4.4 on page 54 discusses the allowed and recommended usage of these commands and some of the normal replies to these commands.

7. Not supported in DRDA Remote Unit of Work.

Table 4-1 DDM Commands Used in DRDA Flows

DDM Command	Description
ACCRDB	Access Relational Database—establishes a path to a named relational database
BGNBND	Begin Bind—starts the process of binding a package into a relational database
BNDSQLSTT	Bind SQL Statement—binds an SQL statement to a package
ENDBND	End Bind—indicates that no more Bind commands will be sent and the package is now complete
DRPPKG	Drop Package—deletes a named package from a relational database
REBIND (optional)	Rebind—rebinds an existing package into the same relational database
PRPSQLSTT	Prepare SQL Statement—binds, dynamically, a single SQL statement to a section number in an existing package in a relational database
EXCSQLSTT	Execute SQL Statement—executes a previously bound SQL statement
EXCSQLIMM	Execute SQL Statement Immediate—executes the single SQL statement sent with the command
DSCSQLSTT	Describe SQL Statement—requests definitions of either the columns of the result table of a prepared/bound statement and the names and labels of those columns or to obtain definitions of the input parameters of a prepared statement
DSCRDBTBL (optional)	Describe Table Statement—describes the columns of a table and the names and labels of those columns
OPNQRY	Open Query—requests start of Query process (corresponds to a DCL CURSOR, OPEN, and possibly multiple Fetches)
CNTQRY	Continue Query—resumes a Query that was interrupted
CLSQRY	Close Query—terminates a Query (corresponds to a CLOSE)
RDBCMM	Commit Transaction—commits the current unit of work
RDBRLLBCK	Rollback Transaction—rolls back (backs out) the current unit of work
EXCSQLSET (optional)	Propagate special registers

The commands that the SQLAM handles do not define SQL, but several DDM commands carry SQL statements. The SQLAM recognizes SQL statements but does not define the syntax and semantics of those statements.

4.3.1.12 Relational Database Manager

A relational database manager (RDB) controls the storage and integrity of data in a relational database. DRDA provides no command protocol or structure for relational database managers. The relational database manager is the local interface for the SQLAM in the application server. Neither DRDA nor DDM defines the interface to the relational database manager.

The relational database is responsible for performing any necessary character conversions for data received from the SQLAM.

For detailed definitions and descriptions of the functions the relational database managers perform, see the product documentation for relational database management products.

4.3.1.13 CCSID Manager

The CCSIDMGR allows the specification of a single-byte character set CCSID to be associated with character typed parameters on DDM command and DDM reply messages. The CCSID manager level of the application requester is sent on the EXCSAT command and specifies the CCSID that the application requester will use when sending character command parameters. The application server will return its CCSID manager level on EXCSATRD specifying the CCSID the application server intends to use for character reply message parameters.

Support for the CCSID manager is optional. The following CCSIDs are required when supporting the CCSID manager: 500, 819, and 850. Other CCSIDs are optional. The following CCSID values cannot be sent by the application requester: 65535 and 0. The application server can reply using the following values:

0	The CCSIDMGR is not supported.
65535	The CCSIDMGR is supported but the CCSID value sent by the application requester is not supported.
value	The CCSID of the application server. This value must be one of the required CCSIDs if the application requester sent one of the required CCSIDs. This is to guarantee the application requester and application server will communicate if the application requester is only capable of supporting the required CCSIDs.

If the CCSIDMGR is not supported, the default value is 500.

4.3.2 The DRDA Processing Model Flow

Figure 4-1 on page 47 illustrates DRDA's usage of DDM. It relates distributed relational database management processing to the models described in the DDM Reference. The following discussion relates the terminology and concepts of DRDA to those of the DDM documentation through this illustration.

The sync point manager[8] in Figure 4-1 on page 47 is only used if the local operating environment for the application requester and application server support sync point managers.

8. The sync point manager is only supported in Distributed Unit of Work.

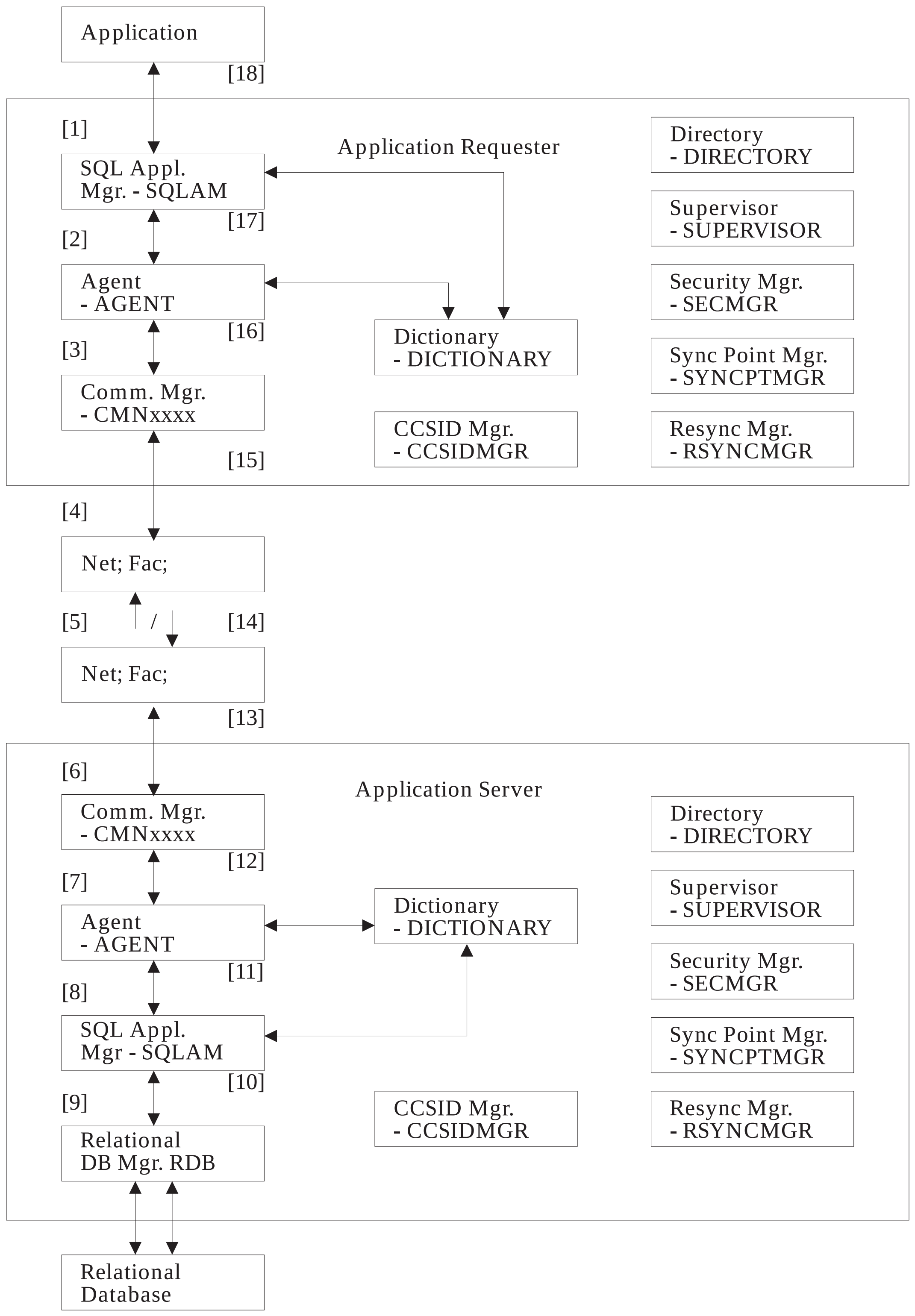


Figure 4-1 DRDA Processing Model

Note: The DDM Reference discusses in detail each of the managers represented in Figure 4-1 on page 47. See the term referenced in the box (for example, SQLAM in the SQL Application Manager box) for the detailed DDM description.

The individual products, not DDM or DRDA, define the interfaces (syntax, semantics) between any of the managers or other entities shown in Figure 4-1 on page 47.

DDM also groups managers into servers. In Figure 4-1 on page 47, all of the managers in the source system form a server, which DRDA calls the application requester. In the target system, all of the managers form a server, which DRDA calls the application server.

The numbered paragraphs that follow correspond to the numbers in Figure 4-1 on page 47 and are a description of the interaction between each of the entities in the figure. The figure contains all of the model entities that would exist for an application to access a relational database using DRDA. This example assumes that the application's SQL statements and associated host variables were previously bound to the remote relational database. Some managers initiate/maintain the connection, others are used by other managers for specific kinds of services, and the rest are an integral part of the path between the application and the relational database.

1 The application contains some set of SQL statements that have been previously bound to the remote relational database. The source code for the application is transparent to the location of the relational database to which it is bound. This transparency is achieved when the application uses ISO Database Language SQL. If the application uses SQL that is not part of ISO SQL, some loss of location or relational database manager transparency can result.

The SQL application manager (SQLAM) represents the remote relational database to the application. The entities that are between the application and the relational database provide the transparency between the differences in hardware architectures, operating systems, and relational database management products.

An application calls the SQL application manager whenever the application requests services through the SQL interface. Neither DDM nor DRDA defines this interface, or set of interfaces, even though some portions of the DDM commands in DRDA resemble parts (or all) of these interfaces.

Calls to this interface are generated by the program preparation process and can be different in each implementation. This interface is composed of input variables from the application, SQL statements or identifiers of previously bound SQL statements, and an area that is to be used to return completion information to the application. These parameters and their associated values vary across the different implementations of SQL application managers (SQLAMs).

In any implementation of an application requester, any manager in the model can access the security manager to determine the user's authority to access any of the local resources (for example, the communications facilities). Neither the DDM architecture nor DRDA completely defines the interface to the security manager.

2 The SQL application manager (SQLAM), when called, processes the request by checking the parameters, translating the valid requests, and packaging the requests into zero[9] or more DDM commands and associated parameters and command data.

9. It is possible to have situations where no DDM commands would be generated for an application's request. In this case, the application requester would usually provide a proper response to the application. An example would be the application attempting another FETCH operation after the cursor has been closed.

The SQLAM uses functions modeled in the dictionary manager to determine the code points (each DDM command, command parameter, data object, reply message, and reply message parameter has a unique code point) to be used in the DDM commands.

As it constructs the commands, it also does any required data representation conversion of command parameter values. Unless overridden by the DDM CCSID manager (CCSIDMGR),[10] the character parameter values are sent in CCSID 500.

Command data objects are constructed using code points that either imply the format of the data or explicitly contain a description of the data that follows. No data representation conversion is required for command data objects because the target SQLAM is the receiver of the data object and, therefore, is responsible for any conversion required.

It then passes these DDM commands to the source agent.

3 The agent receives the DDM commands, parameters, and data objects from the SQL application manager and routes them to the DDM communications manager. It also keeps track of each individual command, as it does for all commands passed to the DDM communications manager, until a reply to the command is received.

4 The DDM communications manager (that is, CMNAPPC, CMNSYNCPT, CMNTCPIP, and so on)[11] receives the DDM command and creates a DDM data stream structure that contains the command.

For each DDM command, a request data stream structure (RQSDSS) is created and the command placed in it. A request correlation identifier is generated and placed in the data stream to be used to associate this request with request data, replies to the request, and data returned for the request. The request correlation identifier is returned to the agent.

For each DDM command data object received, an object data stream structure (OBJDSS) is created, and the command data object is placed in it. Each OBJDSS can contain multiple command data objects, but they must all be part of the same command. The request correlation identifier of the associated RQSDSS is also placed in OBJDSS. The agent provided the request correlation identifier.

The DDM communications manager then invokes the local system's network facilities.

5 The network facilities of the application requester send the commands, parameters, and data objects, as data, through the network to their counterpart network facilities on the application server system. The network facilities in the two systems are responsible for keeping the network connection intact between the application requester and application server as well as for error recovery for the network facilities.

6 Upon receipt of the data at the application server's network facilities, the DDM communications manager invokes the local network facilities to receive the data.

10. This CCSIDMGR is not supported in DRDA Remote Unit of Work.

11. The communications manager at both ends of the communications must match. For example:

- If the network connection is a LU 6.2 protected conversation, CMNSYNCPT is used.
- If the network connection is a LU 6.2 unprotected conversation, CMNAPPC is used.
- If the network connection is a TCP/IP connection, CMNTCPIP is used.

In any implementation of an application server, any manager that is modeled, must access the security manager to determine the user's authority to access any of the local resources (for example, the relational database). Neither the DDM architecture nor DRDA completely defines the interface to the security manager.

7 The DDM communications manager decomposes the data stream it received as data. It breaks the data stream up into the correct commands, parameters, and data objects and passes them on to the agent along with the request correlation identifier. The commands, parameters, data objects, and correlation identifier are the exact ones that the application requester's DDM communications manager sent.

8 The agent receives the information from the communications manager and validates the target parameter of the command as well as any other command parameter and value code points in them. If any errors are found, reply messages are created and returned to the DDM communications manager.

If the command appears to be valid, it is packaged with all data objects (if any) for the same command (all with the same request correlation identifier) and passed to the SQL application manager for processing.

9 The SQL application manager (SQLAM) accepts the commands, parameters, and data objects from the agent and transforms them into one or more calls to the relational database manager it supports. DRDA does not define the interfaces of the relational database manager.

The SQLAM is responsible for transforming any descriptors in the command data it received to the interfaces that the relational database manager expects. It is also responsible for converting numeric data from the application requester's representation to the application server's representation when these are different. Because the relational database handles tagged character data,[12] the application server passes this data directly to the relational database without conversion.

The SQLAM also carries out any required data representation conversion of command parameter values. Unless overridden by the DDM CCSID manager (CCSIDMGR), the character parameter values are received in CCSID 500.

10 The relational database manager receives the requests from the SQLAM, translates character data if appropriate, and processes the requests against the relational database that was indicated when the application server was established for this application's use.

It generates any answer set data, return codes, and error data, according to its product specification, and returns these to the SQLAM. DRDA does not define these responses and associated data.

11 The SQL application manager (SQLAM) transforms the responses and associated data into DDM command reply data objects and reply messages. If any of the reply data objects require explicit descriptions, then the SQLAM creates them, reversing the process of step 9. No data representation conversion is required because the source SQLAM is the receiver of the data object and, therefore, is responsible for all data representation conversions on the reply data objects.

12. Tagged character data is data with coded character set identifiers (CCSIDs) associated with it.

The target SQLAM sends any reply data objects and reply messages on to the agent. The SQLAM also performs data representation conversion as required for reply message parameter values that are sent. Unless overridden by the DDM CCSID manager (CCSIDMGR), the character parameter values are sent and received in CCSID 500.

12 The agent receives the reply data objects and reply messages from the SQLAM and returns them with the correct request correlation identifier to the DDM communications manager.

13 The DDM communications manager receives the command reply data object or reply message from the agent and creates a DDM data stream structure that contains the reply.

For each reply message (if any), a reply data stream structure (RPYDSS) is created and the reply message placed in it. Each RPYDSS can contain multiple reply messages, but they must all correspond to the same request. The request correlation identifier of the original request is also placed in the data stream to be used to associate this reply message with that request.

For each reply data object (if any), an object data stream structure (OBJDSS) is created and the reply data object placed in it. Each OBJDSS can contain multiple reply data objects, but they must all correspond to the same request. The request correlation identifier of the original request is also placed in the data stream to be used to associate this reply data object with that request.

The DDM communications manager then invokes the local system's network facilities to pass the reply back to the application requester system.

14 The application server's network facilities send the replies, as data, through the network back to its counterpart network facilities on the application requester system.

15 Upon receipt of the data from the application server's network facilities, the communications manager invokes the local network facilities to receive the data.

16 The DDM communications manager decomposes the data stream it received as data. Then it breaks the stream up into the correct reply data objects and reply messages and passes them on to the agent along with the request correlation identifier.

17 The agent passes the reply data objects and reply messages to the SQLAM.

18 The SQLAM converts any DDM architecture required format data representation on reply message parameters to the SQLAM required representation.

The SQLAM is responsible for transforming descriptors it received to the interfaces expected by the application that made the request. Neither DDM architecture or DRDA defines the actual form and format of the data/response to the application.

The SQL application manager (SQLAM) does any data representation conversion on reply data objects to the representation that the application requires from the representation of the application server. In this case, the SQLAM does both numeric and character conversions.

4.3.3 Product-Unique Extensions

In a DRDA environment, which contains multiple operating environment products and/or multiple relational database management products, the participating DRDA implementations must use only those code points described in the DDM Reference, according to the limitations and rules that DRDA describes.

Each DRDA implementing product is required to implement the DDM EXCSAT and ACCRDB commands (for application requesters) and EXCSATRD and ACCRDBRM replies (for application servers) as described in Section 4.4.1 on page 54. Once the application requester and application server have been introduced by this architected exchange and recognize each other as a specific product pair, they can use, in addition to the DDM commands listed in Table 4-1 on page 45, product-unique code points in further exchanges between the application requester and application server.

DRDA implementing products cannot, however, implement unique extensions that are in conflict with DRDA. For example, a product unique extension in a product that allows the start of a new bind (package create) before a previous bind process had been completed would not be allowed because it contradicts the DRDA rules for DRDA BIND flows.

The DDM Reference discusses additional detail on product extensions, specifically under the DDM terms EXTENSIONS, CODPNT, CODPNTDR, and SUBSETS.

4.3.4 Diagnostic and Problem Determination Support in DRDA

The DRDA-defined flows contain facilities that are available to end users and customer support organizations to assist in performing problem determination procedures.

The network facilities that support the application requester and application server might provide many facilities that furnish diagnostics and do problem determination procedures. These facilities can be used to supplement the DRDA facilities. For example, the LU 6.2 network facilities are described in *SNA LU 6.2 Reference: Peer Protocols* (SC31-6808, IBM). Additional facilities may be provided in specific communications product implementations that enhance the problem determination capabilities in a particular system, application server, or application requester.

The application requester and the application server exchange information that is intended to identify the application requester and the application server to each other when the remote relational database is accessed. This information includes the products being used, the levels of those products, the operating systems being used, the name of the system each is executing in, and the name of the execution thread in each system.

To the extent that this information is passed using values that end users and service personnel easily understand/recognize, the tasks associated with problem determination and diagnostic handling are simplified. Therefore, each implementing product is required to accurately reflect its environment by assigning values in the parameters that carry this information as character strings that are encoded using Coded Character Set Identifier (CCSID) 500, unless overridden by the DDM CCSID manager (for additional information see the CDRA Reference). Each application requester and application server is required to store (for potential use later) the information the other has provided.

Character Data Representation Architecture (CDRA) defines CCSID values to identify the code points used to represent characters, and the character data conversion of these code points, as needed, to preserve the characters and their meanings.

Every command that an application requester sends to an application server has a number of architected reply messages that the application server can return to the application requester.

Each reply message contains data that can be used in problem determination procedures. The return of a particular reply message can, by itself, facilitate problem determination procedures.

Each reply message also contains a severity code and, potentially, server diagnostics. The severity codes values (see DDM term *svrcod*) are in the description of each of the reply messages. The application server returns the data in the server diagnostic information field (see DDM term *srvdgn*). The application requester handles the data only as a byte string. The application requester must store it in its entirety for potential use later. The application server implementing product provides the definition of the data contained in the server diagnostic information field. This data can differ across the different application server implementations.

In some cases, the application requester can use information in an SQLCA that one of the architected reply data objects returned to provide facilities for problem determination procedures. This type of support can differ across the different application requester product implementations.

4.4 DDM Commands and Replies

The following sections describe the DDM commands and command replies that flow in typical scenarios involving an application requester (AR) and an application server (AS).[13] These flows are equivalent between an application server (AS) and a database server (DS) but are not specifically described. The terms application requester and application server can be interchanged with an application server and a database server unless specifically identified in the flow.

These sections contain flow diagrams and descriptions that show the normal or successful case, and do not cover all possible error conditions or obscure usages. Errors and replies are generalized rather than elaborated upon. Complete details of the commands, parameters, command data, reply data, and error conditions / messages are available in the DDM Reference.

The usage of the underlying communications facilities is presented only when it is an integral part of the DRDA processing.

4.4.1 Accessing a Remote Relational Database Manager

Figure 4-2 on page 55 indicates the DDM commands and replies that flow in the normal process of establishing a connection for remote processing of DRDA requests. This set of flows establishes a connection from an application requester to a remote application server. After the application requester establishes the connection and until the connection has been terminated, either normally or abnormally, the DRDA flows can use the connection.

13. It is possible to intermix DDM commands that are not part of DRDA in with the command flows that this section discusses. However, these commands and their potential interaction are not discussed in this document.

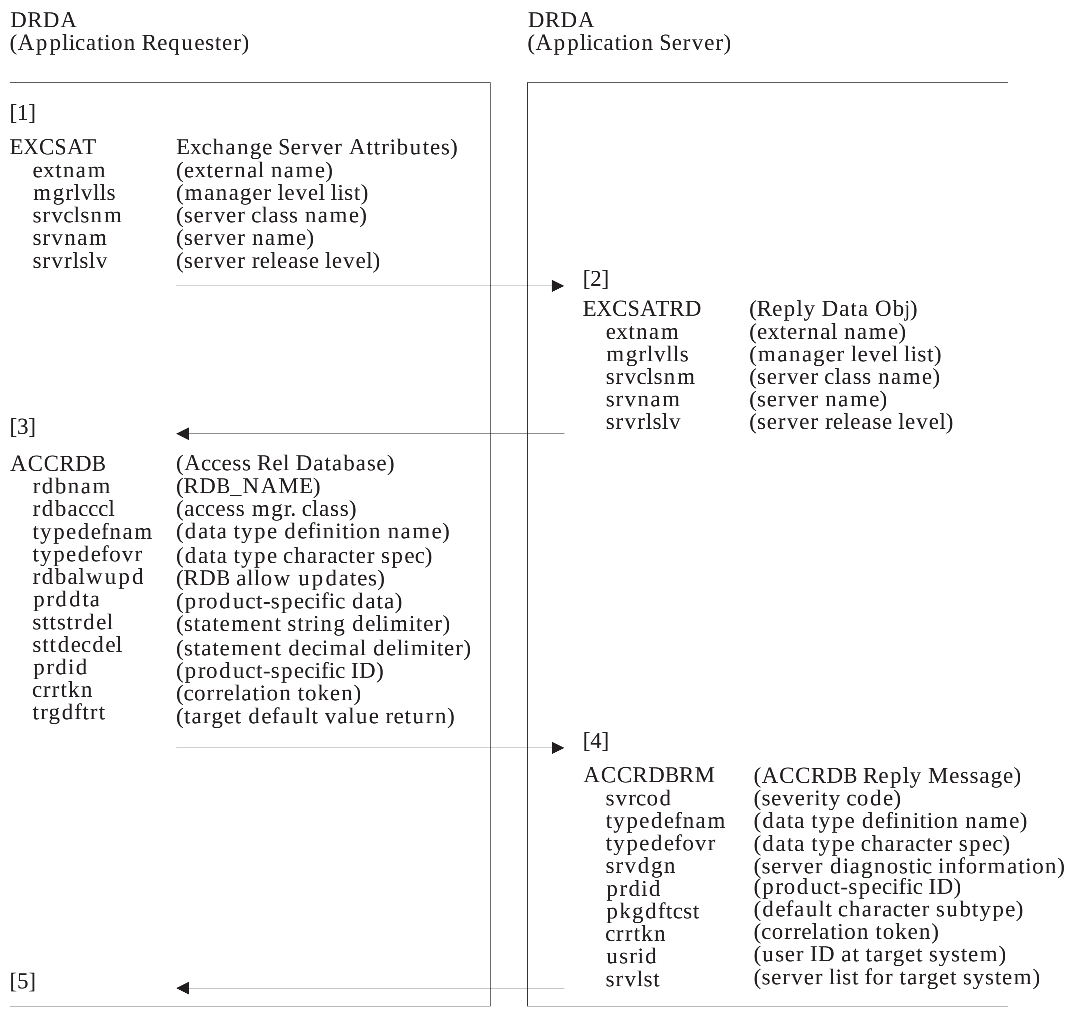

DRDA
(Application Requester)
DRDA
(Application Server)
[1]
EXCSAT Exchange Server Attributes)
extnam (external name)
mgrlvlls (manager level list)
srvclsnm (server class name)
srvnam (server name)
srvrlslv (server release level)
[2]
EXCSATRD (Reply Data Obj)
extnam (external name)
mgrlvlls (manager level list)
srvclsnm (server class name)
srvnam (server name)
srvrlslv (server release level)
[3]
ACCRDB (Access Rel Database)
rdbnam (RDB_NAME)
rdbacccl (access mgr. class)
typedefnam (data type definition name)
typedefovr (data type character spec)
rdbalwupd (RDB allow updates)
prddta (product-specific data)
sttstrdel (statement string delimiter)
sttdecdel (statement decimal delimiter)
prdid (product-specific ID)
crrtkn (correlation token)
trgdftrt (target default value return)
[4]
ACCRDBRM (ACCRDB Reply Message)
svrcod (severity code)
typedefnam (data type definition name)
typedefovr (data type character spec)
srvdgn (server diagnostic information)
prdid (product-specific ID)
pkgdftcst (default character subtype)
crrtkn (correlation token)
usrid (user ID at target system)
srvlst (server list for target system)
[5]

Figure 4-2 DRDA Flows to Establish a Connection to a Remote Database Manager

The following is a brief description of some of the parameters for the DDM commands. The DDM Reference provides a detailed description of the parameters.

1 The application requester makes a connection which includes establishing a network connection (described in Part 3, Network Protocols) with the application server. Part 3 discusses the protocols and commands used to establish a network connection for each of the DRDA-supported network protocols.

After establishing the network connection, the application requester describes itself and the types of services that it desires from the application server.

The application requester builds an Exchange Server Attributes (EXCSAT) command, identifying what product it is, what release and modification level it is at, and what this application requester is known as in its environment. EXCSAT also lists the level of the communications manager, the agent, the SQLAM, the relational database manager, and any other resource managers that the application requester requires in the manager level list. The application requester then sends the command to the application server.

The following example of the EXCSAT command shows the parameters and values that establish the connection between an application requester and an application server.

```
EXCSAT(
       extnam("015190/JOB39/WSDD1234")
       mgrlvlls(
                mgrlvl(AGENT,5)
                mgrlvl(SECMGR,5)
                mgrlvl(CMNTCPIP,5)
                mgrlvl(SYNCPTMGR,5)
                mgrlvl(SQLAM,5)
                mgrlvl(CCSIDMGR,500)
                mgrlvl(RDB,3))
       srvclsnm("QAS")
       srvnam("RCHOLDB")
       srvrlslv("QSQ02011"))
```

- The *extnam* is the name of a job, task, or process that the application requester services. It is used for diagnostic/logging purposes. In this example, it is the name of the job that contains the execution of the application that is invoking application requester functions on the OS/400 system.

  Note: This parameter is required and must contain the name of the application requester's execution thread in its operating environment. It must be a name that an observer of the operating environment can easily associate with its execution.

- The *mgrlvlls* is the minimum list necessary to determine that the source and target manager levels are compatible for DRDA functions. The *mgrlvlls* on EXCSAT represent the desired support that the application requester needs on the application server. In this case, a DRDA TCP/IP application requester supports Distributed Unit of Work and stored procedures, so it requests an agent (AGENT) at Level 5, a security manager (SECMGR) at Level 5, a TCP/IP communications manager (CMNTCPIP) at Level 5, a sync point manager (SYNCPTMGR) at Level 5, an SQL application manager (SQLAM) at Level 5, and a relational database manager (RDB) at Level 3.

  If the application requester is a DRDA application requester that does not support Distributed Unit of Work but does support stored procedures, it requests an agent (AGENT) at Level 3, an SQL application manager (SQLAM) at Level 5, a relational database manager (RDB) at Level 3, and an SNA communications manager (CMNAPPC) at Level 3, or a TCP/IP communications manager (CMNTCPIP) at Level 5 which requires a security manager (SECMGR) at Level 5.

  If the application requester is a DRDA Remote Unit of Work application requester, it requests an agent (AGENT) at Level 4, an SQL application manager (SQLAM) at Level 3, a relational database manager (RDB) at Level 3, and a DDM communications manager (CMNAPPC) at Level 3, or a TCP/IP communications manager (CMNTCPIP) at Level 5 which requires a security manager (SECMGR) at Level 5.

  The value specified for the CCSID manager (CCSIDMGR) indicates what CCSID the application requester uses when sending character typed command parameters. In this example, the application requester is sending character command parameters in CCSID 500. If the application server supports the CCSID manager, the CCSID used by the application server for character reply parameters is returned by the application server on EXCSATRD. In this example, the CCSID sent by the application requester is one of the required CCSIDs (500, 819, or 850) and, assuming the application server supports the CCSID manager, the application server must return a required CCSID.

Note: The *mgrlvlls* parameter is required and must include the AGENT, SQLAM, RDB, and a communication manager. The CCSID manager (CCSIDMGR) is optional and is included as an example of the negotiation for this function.

The table below, Table 4-2, summarizes the types of read and write access that can be accomplished based on the *mgrlvlls* specified on EXCSAT and EXCSATRD.[14]

Table 4-2 Access by the Minimum MGRLVLLS Parameter of EXCSAT and EXCSATRD

AGENT	3	4	4	5
SQLAM	3	4	4	4
CMNAPPC	3	3		
CMNTCPIP	5	5		5
CMNSYNCPT			4	
SYNCPTMGR			4	5
Remote Unit of Work with Single-RDB Access	yes	no	no	no
Distributed Unit of Work with Multi-RDB Read and Single-RDB Write	no	yes	no	no
Distributed Unit of Work with Multi-RDB Read and Multi-RDB Write	no	no	yes	yes

Note: A blank entry indicates that the DDM manager is not applicable for the requested level. CMNAPPC, CMNTCPIP, and CMNSYNCPT are mutually exclusive. If CMNTCPIP is specified, SECMGR at Level 5 must be specified.

- The *srvclsnm* identifies the application requester. In this case, 'QAS' indicates that the application requester is the DB2 for OS/400 product. Server class names are assigned for each product involved in DRDA. Server class names for products involved in DRDA can be found at *http://www.opengroup.org/tech/projects/drda*.

  Note: The *srvclsnm* term in the DDM Reference defines all the values that have been assigned for *srvclsnm*. This parameter is required.

- The *srvnam* is the name of the application requester. This is not the name of the end user but of the server itself. It is for diagnostic/logging purposes. In this case, the name corresponds to the system name of the OS/400 in which the application requester is executing.

  Note: This parameter is required and must contain the name of the application requester's system identifier in the network of application requester and application server. It must be a name that an observer of the network containing the system can easily associate with the system in which the application requester is executing.

14. The DRDA support for stored procedures, new bind options, and server list require SQLAM Level 5.

- The *srvrlslv* is the current release level of the application requester and is for diagnostic/logging purposes. Because this level applies to all managers at the application requester site (or at the application server site on the EXCSATRD), it cannot be specific to the DRDA service provider. It is considered optional in DRDA. DRDA has provided the *prdid* parameter on ACCRDB and ACCRDBRM to identify the release levels of the application requester and the application server.

  In this example,[15] the application requester has identified itself as the DB2 for OS/400 product, running at Version 2, Release 1, Modification Level 1.

2 The application server receives the EXCSAT command, builds the reply data object, Exchange Server Attributes Reply Data Object (EXCSATRD), and stores the data received on the EXCSAT command for potential diagnostic/service uses. The application server does not verify or check values in *extnam*, *srvclsnm*, *srvnam*, or *srvrlslv*.

The application server places the levels of all managers requested in the reply data along with the set of information that describes the application server and its environment.

The following example shows the parameters and values that an EXCSATRD reply data object uses.

```
EXCSATRD(
         extnam("SYSD9876")
         mgrlvlls(
                  mgrlvl(AGENT,5)
                  mgrlvl(SECMGR,5)
                  mgrlvl(CMNTCPIP,5)
                  mgrlvl(SYNCPTMGR,5)
                  mgrlvl(SQLAM,5)
                  mgrlvl(CCSIDMGR,500)
                  mgrlvl(RDB,3))
         srvclsnm("QDB2")
         srvnam("STLDB2A1")
         srvrlslv("DSN03010"))
```

- The *extnam* is the name of a job, task, or process (in this example SYSD9876) that the application server is running under. It is for diagnostic/logging purposes. In this example, it is the name of the job that contains the execution of the application server functions and DB2 for MVS. The rules for assigning a value to this parameter are the same as in the application requester.

- The application server returns the *mgrlvlls*, which indicate the levels of the requested (and only those requested) managers that the application server is capable of supporting. The values in this example (an agent at Level 5; a DDM communications manager, CMNTCPIP at Level 5; a security manager at Level 5; an SQL application manager at Level 5; and a relational database manager at Level 3) indicate that the application requester and application server can communicate with DRDA flows, using DDM commands, at the level the application requester

15. The version, release, and modification levels defined in this example for *srvrlslv* are for example purposes only and do not represent actual product levels.

requested.

The application server returns the CCSID in which it sends character reply parameters using the CCSID manager level. This CCSID must be one of the required CCSIDs (500, 819, or 850) if the application server supports the CCSID manager and a required CCSID was received on EXCSAT. If the CCSID *mgrlvl* sent by the application requester is not a required CCSID *mgrlvl* and the application server cannot accept the CCSID mgrlvl, the application server returns the EXCSATRD with a −1 CCSID *mgrlvl* specification.

If the application requester cannot accept the CCSID *mgrlvl*[16] received from the application server the conversation is terminated.

- The *srvclsnm*, *srvnam*, and *srvrlslv* have the same semantics as their counterparts on the EXCSAT command and the rules for assigning values to these parameters are the same as those at the application requester.

  In this example, the application server has identified itself as a DB2 for OS/390 product, running at Version 5, Release 1, Modification Level 0 and executing on system STLDB2A1.

3 When the application requester receives the reply data from the EXCSAT command, it determines if the level of support that a target relational database manager can provide is sufficient to meet its needs. If not, the application requester terminates the conversation and returns an exception to the application. The application requester does not verify or check the values in the *extnam*, *srvclsnm*, *srvnam*, or *srvrlslv* except those that are needed to determine if the application requester and application server are a specific product pair (see Section 4.3.3 on page 52 for details).

These returned values are stored for potential diagnostic/service uses. If the levels of support are sufficient, the application requester creates the Access Relational Database (ACCRDB) command and sends it to the application server.

- *rdbnam* contains the name of the desired relational database.
- The *rdbacccl* parameter indicates that the process will use DRDA flows for processing a user application's SQL requests.
- The *typdefnam* parameter indicates the data type to data representation mapping definitions, which the application requester will use. Refer to Table 5-20 on page 251 for details.
- The *typdefovr* parameter indicates the desired Coded Character Set Identifiers (CCSIDs) in the identified data type to data representation mapping definitions. Refer to Table 5-20 on page 251 for details.
- The *rdbalwupd* parameter specifies whether the application server should allow updates to occur. An update operation is defined as a change to an object at the relational database, such that the change to the object is under commit/rollback control of the work that the application requester initiates.

  When the application requester specifies that no updates are allowed, the application server must enforce this specification and, in addition, must not allow

16. The *mgrlvlls* defined in this example are for example purposes only and do not imply the product in the example provides support for the specified managers or levels.

the execution of a commit or rollback that the DDM command EXCSQLIMM or EXCSQLSTT requested.

- The *prddta* parameter specifies product-specific information[17] that the application requester conveys to the application server if the *srvclsnm* of the target is not known at the time ACCRDB is issued and the application requester must convey such product-specific information.
- The *sttstrdel* and *sttdecdel* parameters respectively specify the statement string delimiter and decimal delimiter for dynamic SQL.
- The *prdid* is the current release level of the application requester and is for diagnostic/logging purposes. The *prdid* should be unique amongst the DRDA implementers.

  This parameter is required and must be of the form PPPVVRRM where:

ppp	A three-character product identifier. Refer to *http://www.opengroup.org/tech/projects/drda* for the current list of product identifiers.
vv	*dd*, where *d* is an integer and $0 \leq d \leq 9$ (for single digit version numbers, pad on the left with 0).
RR	*dd*, where *d* is an integer and $0 \leq d \leq 9$ (for single digit release numbers, pad on the left with 0; 00 means no release number associated with the level of the product).
M	*d*, where *d* is an integer and $0 \leq d \leq 9$ (0 means no modification level associated with the level of the product).

- The *trgdftrt* parameter is optionally set to TRUE requesting the application server to return its system defaults values in ACCRDBRM.
- The *crrtkn* parameter contains a correlation token. This parameter is optional in DRDA Remote Unit of Work, and is required in DRDA Distributed Unit of Work and DRDA Level 3. See Section 11.2.2.2 on page 353 for details on setting the value of this token.

The application requester then sends the command to the application server.

4 When the application server receives the ACCRDB command, it verifies the command and, assuming everything is acceptable, establishes a connection to the relational database manager of the relational database requested, through a new instance of the SQLAM.

It then generates an Access RDB complete reply message (ACCRDBRM) that indicates a normal completion of this request and provides the application requester with additional information about the application server.

- The *typdefnam* parameter indicates the type to representation mapping definition that the application server uses. Refer to Chapter 5 on page 137 for details.

17. An application server must ignore product-specific information unless received from a like application requester.

The application server also indicates its desired CCSIDs in the identified data type to data representation mapping definitions in the *typdefovr* parameter. Refer to Chapter 5 on page 137 for details.

- The application server returns the *prdid* parameter, specifying the current release level of the application server.
- If the *trgdftrt* parameter on ACCRDB was set to TRUE, the character subtype system default is returned in *pkgdftcst* and the user ID is returned in *usrid*.
- The *crrtkn* parameter contains a correlation token. See Part 3, Network Protocols for the format and settings of the token value in the specific network environments. The *crrtkn* parameter is sent on the ACCRDBRM only if the *crrtkn* parameter is not received on the ACCRDB.
- The *srvlst* parameter contains a weighted list of network addresses that can be used to access the RDB. The list can be used by the requester to work load balance future connections. This parameter is new in DRDA Level 3. Details of the server list and examples are in the DDM references.

If the application server finds any abnormal conditions, it would generate and return a DDM reply message, indicating the error condition and supporting diagnostic information. The application server also would not complete the connection to the relational database manager and the relational database.

5 The application server sends the ACCRDBRM or another DDM reply message to the application requester. The application requester then determines if there is a proper connection established to the requested relational database. If the *typdefnam* or *typdefovr* parameters on an ACCRDBRM cannot be supported or the DDM reply message indicates another error, then the error is indicated to the application in the appropriate way. The errors will be indicated in the SQLCA for SQL errors or according to rules of the specific product for non-SQL errors, and the conversation will be deallocated.

If the ACCRDBRM indicates no problems have been discovered, the application requester will continue (either return to the application or begin working with the connected application server) normal processing.

The application and the remote relational database have now completed the connection. SQL requests, through defined flows, can now be executed in the application server environment on behalf of the application in the application requester environment.

4.4.2 DRDA Security Flows

This section describes the DDM commands and replies for flowing security information in DRDA when not using the underlying communications manager for authentication. DRDA provides flows for the security mechanisms listed in Table 4-3 on page 62. (For example, DCE security, userid only, userid and password, and so on.)

4.4.2.1 Identification and Authentication Security Flows

The flows in this section indicate the DDM commands and replies that flow in the normal process of establishing a connection while using DRDA-defined flows to perform identification and authentication using various security mechanisms. The actual security mechanism that is in use is dependent on the results of the negotiation during ACCSEC/ACCSECRD flows. The security mechanism in use also defines the parameter values during SECCHK/SECCHKRM flows.

Table 4-3 Security Mechanism to *secmec* Value Mapping

Security Mechanism	Secmec Value
DCE	*dcesec*
Userid and password	*usridpwd*
Userid, password, and new password	*usridnwpwd*
Userid only	*usridonl*
Userid and password substitute	*usrsbspwd*
Userid and password encryption	*usrencpwd*

More information on the security mechanisms is available in Chapter 10 on page 343.

Figure 4-3 shows the DCE security flows:

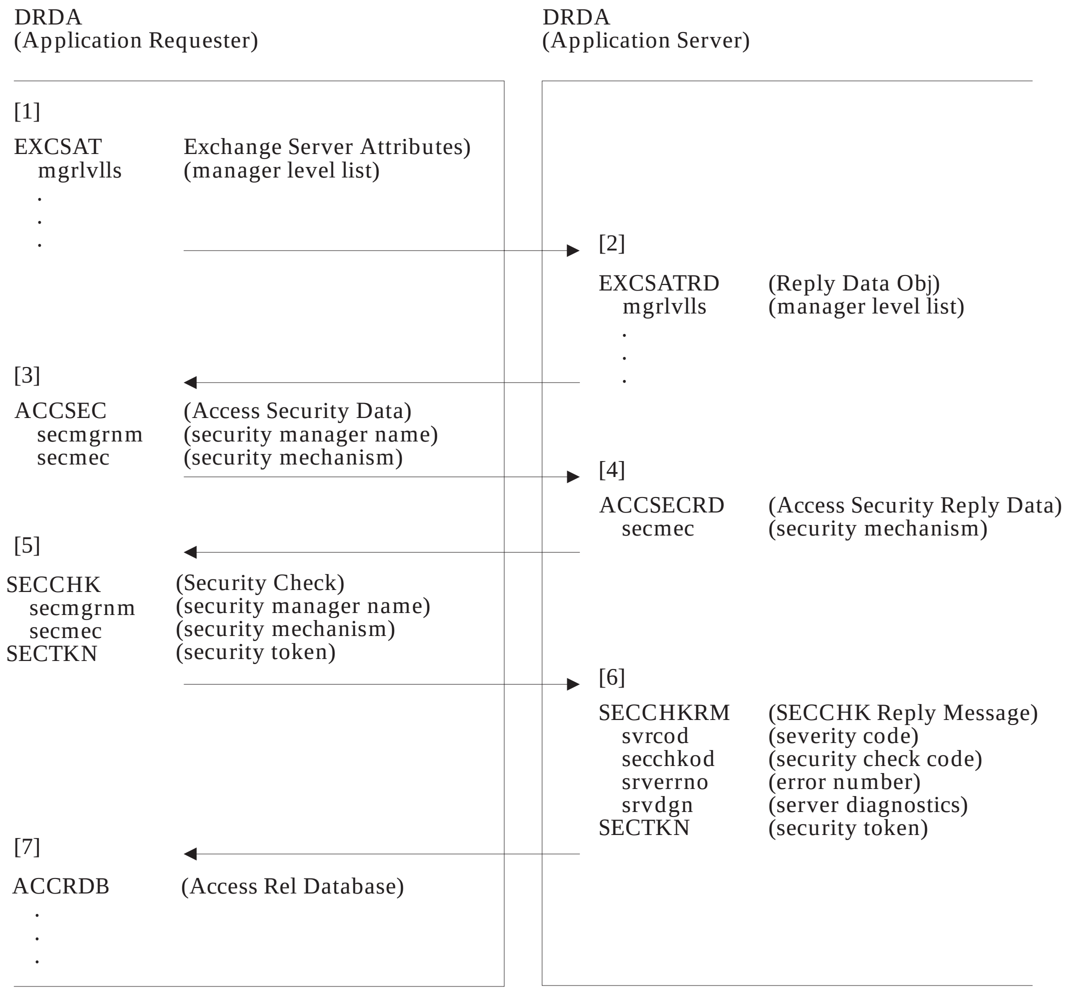

DRDA
(Application Requester)
DRDA
(Application Server)
[1]
EXCSAT Exchange Server Attributes)
mgrlvlls (manager level list)
[2]
EXCSATRD (Reply Data Obj)
mgrlvlls (manager level list)
[3]
ACCSEC (Access Security Data)
secmgrnm (security manager name)
secmec (security mechanism)
[4]
ACCSECRD (Access Security Reply Data)
secmec (security mechanism)
[5]
SECCHK (Security Check)
secmgrnm (security manager name)
secmec (security mechanism)
SECTKN (security token)
[6]
SECCHKRM (SECCHK Reply Message)
svrcod (severity code)
secchkod (security check code)
srverrno (error number)
srvdgn (server diagnostics)
SECTKN (security token)
[7]
ACCRDB (Access Rel Database)

Figure 4-3 DCE Security Flow

The following is a brief description of some of the parameters for the DDM commands. The DDM Reference provides a detailed description of the parameters.

1 The application requester specifies an AGENT at Level 3 and a SECMGR at Level 5 on EXCSAT when requesting the use of DRDA flows for identification and authentication.

The AGENT and SECMGR requires the ACCSEC and SECCHK commands to flow prior to ACCRDB. Neither command can flow after ACCRDB. The other *mgrlvl* values that are required for establishing a connection are described in Section 4.4.1 on page 54.

```
EXCSAT(
       mgrlvlls(
                mgrlvl(AGENT,3)
                mgrlvl(SECMGR,5)
                  .
                  .
                  .))
```

2 The application server receives the EXCSAT command and builds a reply data object with an AGENT at Level 3 and a SECMGR at Level 5 indicating it can operate at that security level. The application server sends the EXCSATRD reply data to the application requester.

```
EXCSATRD(
         mgrlvlls(
                  mgrlvl(AGENT,3)
                  mgrlvl(SECMGR,5)
                    .
                    .
                    .))
```

3 The application requester receives the EXCSATRD reply data which indicates the application server supports an AGENT and SECMGR level that allows negotiation for the type of identification and authentication mechanisms through the ACCSEC command.

The *secmec* parameter indicates the type of security mechanism that will be used. The *secmec* values per security mechanism mapping are defined in Table 4-3 on page 62.

In this example, the application requester requests DCE security so the *dcesec* value is passed in the *secmec* parameter.

4 The application server receives the ACCSEC command. It supports the security mechanism identified in the *secmec* parameter, so the application server reflects the same security mechanism back to the application requester in the *secmec* parameter on the ACCSECRD reply data object.

If the application server does not support the security class specified in the *secmec* parameter on ACCSEC command, the application server returns a list of security mechanism values that it does support in the *secmec* parameter on ACCSECRD reply data object.

5 The application requester receives the ACCSECRD reply data object and calls security services for the mechanism in use, to generate the security token required for security processing. The actual process to generate the token is not specified by DRDA. The Generic Security Services-Application Programming Interface (GSS-API) is a security API for generating a DCE security token. A DRDA implementation might use another interface, but the generated token must be equivalent to the token generated by GSS-API.

If the values received in the *secmec* parameter on ACCSECRD do not match the values sent in the *secmec* parameter on ACCSEC, the application requester either uses one of the security mechanisms received on ACCSECRD or the application requester should drop the connection and return an SQLCA to the application with an SQLSTATE value of X'0A501' indicating a connection could not be established.

The application requester passes the security token in a SECTKN object with the SECCHK command. The *secmec* parameter value identifies the security mechanism in use.

6 The application server receives the SECCHK command and uses the security context information to perform end-user identification and authentication checks.

The actual process to perform the security checks using the security context information is not specified by DRDA. The application server may either process the values itself or it may call a security resource manager interface to process the values.

Assuming authentication is successful, the application server generates a SECCHKRM reply message to return to the application requester. The *secchkcd* parameter identifies the status of the security processing. The SECTKN carries security context information to perform identification and authentication of the application server. There will not be a SECTKN returned for userid and password mechanism and userid only mechanism.

A failure to authenticate the end user or successfully pass the security checks results in breaking the chain if other commands are chained to the SECCHK command. The *svrcod* parameter must contain a value of 8 or greater if the chain is broken.

7 The application requester receives the SECCHKRM reply message. Assuming authentication at the application server is successful, the application requester verifies the security token received in the SECTKN.

Assuming security processing is successful, the application requester sends an ACCRDB command to the application server.

If security processing fails, the application requester might attempt recovery before returning to the application. For example, if the security context information in the security token has expired as indicated by the *secchkcd* value, the application requester could request new security context information to send to the application server. If the error condition is not recoverable, the application requester returns an SQLCA to the application with an SQLSTATE value of X'42505' indicating a security verification failure.

Figure 4-4 on page 65 shows the password encryption or substitution flows:

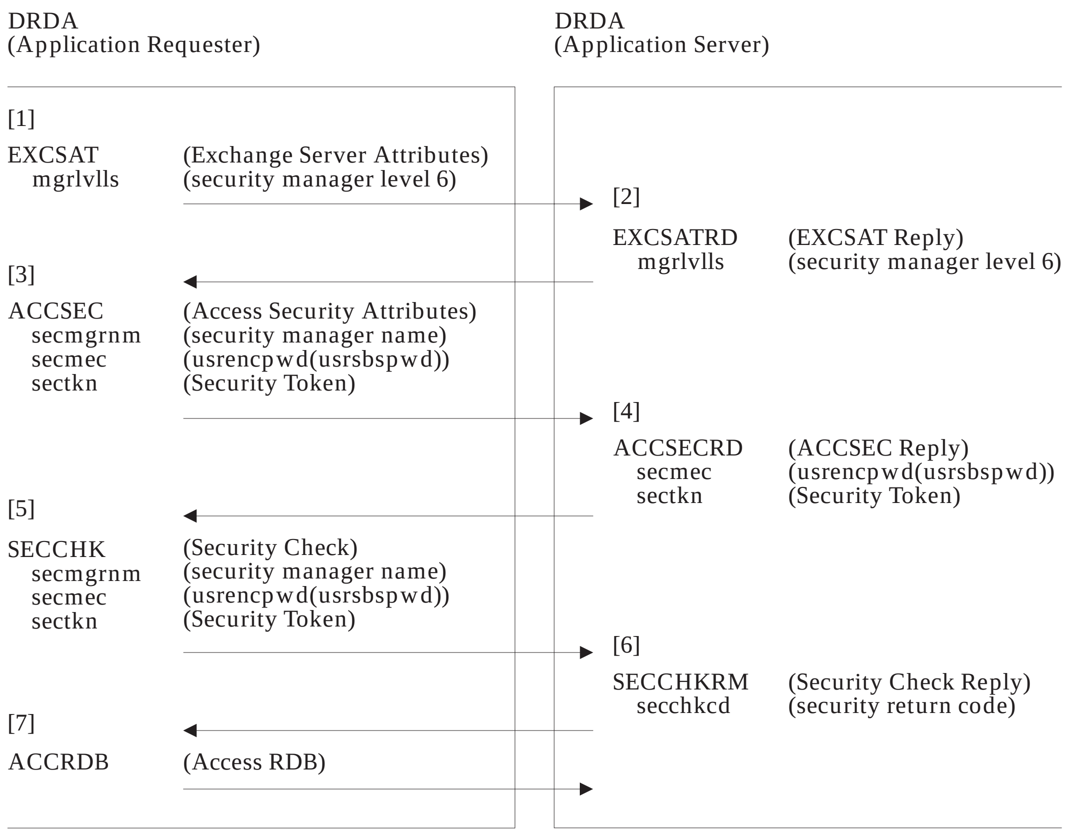

DRDA
(Application Requester)
DRDA
(Application Server)
[1]
EXCSAT (Exchange Server Attributes)
mgrlvlls (security manager level 6)
[2]
EXCSATRD (EXCSAT Reply)
mgrlvlls (security manager level 6)
[3]
ACCSEC (Access Security Attributes)
secmgrnm (security manager name)
secmec (usrencpwd(usrsbspwd))
sectkn (Security Token)
[4]
ACCSECRD (ACCSEC Reply)
secmec (usrencpwd(usrsbspwd))
sectkn (Security Token)
[5]
SECCHK (Security Check)
secmgrnm (security manager name)
secmec (usrencpwd(usrsbspwd))
sectkn (Security Token)
[6]
SECCHKRM (Security Check Reply)
secchkcd (security return code)
[7]
ACCRDB (Access RDB)

Figure 4-4 Password Encryption or Substitution Flow

1 The application requester specifies a SECMGR at Level 6 on the EXCSAT command when requesting the use of DRDA for identification and authentication. SECMGR Level 6 is required to support the SECTKN instance variable on the ACCSEC command and the ACCSECRD reply as well as the new SECCHKCD instance variable on the ACCSECRD reply.

2 The application server processes the EXCSAT command and builds a reply data object. If the new SECTKN instance variable is supported on the ACCSEC command, SECMGR Level 6 is returned; otherwise, only security mechanisms supported by SECMGR Level 5 can be used to authenticate.

3 The application requester processes the reply. If the SECMGR is not at Level 6, the application server does not support the new security flows. If the SECMGR is at Level 6, the ACCSEC command is built with the security mechanism (*secmec*) indicating *usrencpwd* for password encryption or *usrsbspwd* for password substitution. For password substitution, the application requester's password encryption seed is generated. For password encryption, a random large number x is generated and is used to generate the connection key X where X is equal to (g^x mod n)[18]. The connection key or encryption seed is passed in the SECTKN object.

18. Refer to the DDM Reference, USRENCPWD, for a description of the values used to generate the Diffie-Hellman shared secret key and the values used to encrypt and decrypt the password using DES.

4 The application server processes the ACCSEC command. If the application server does not support the requested security mechanism, it returns a list of supported mechanisms; otherwise, the reply data is generated. For password substitute, the application server's encryption seed is generated. For password encryption, a random large number y is generated and is used to generate the connection key Y where Y is equal to (g^y mod n). The encryption seed or connection key is returned in the SECTKN object. The optional SECCHKCD instance variable is returned if and only if an error is detected processing the application requester's SECTKN. Possible errors are wrong seed length or invalid value (a trivial seed).

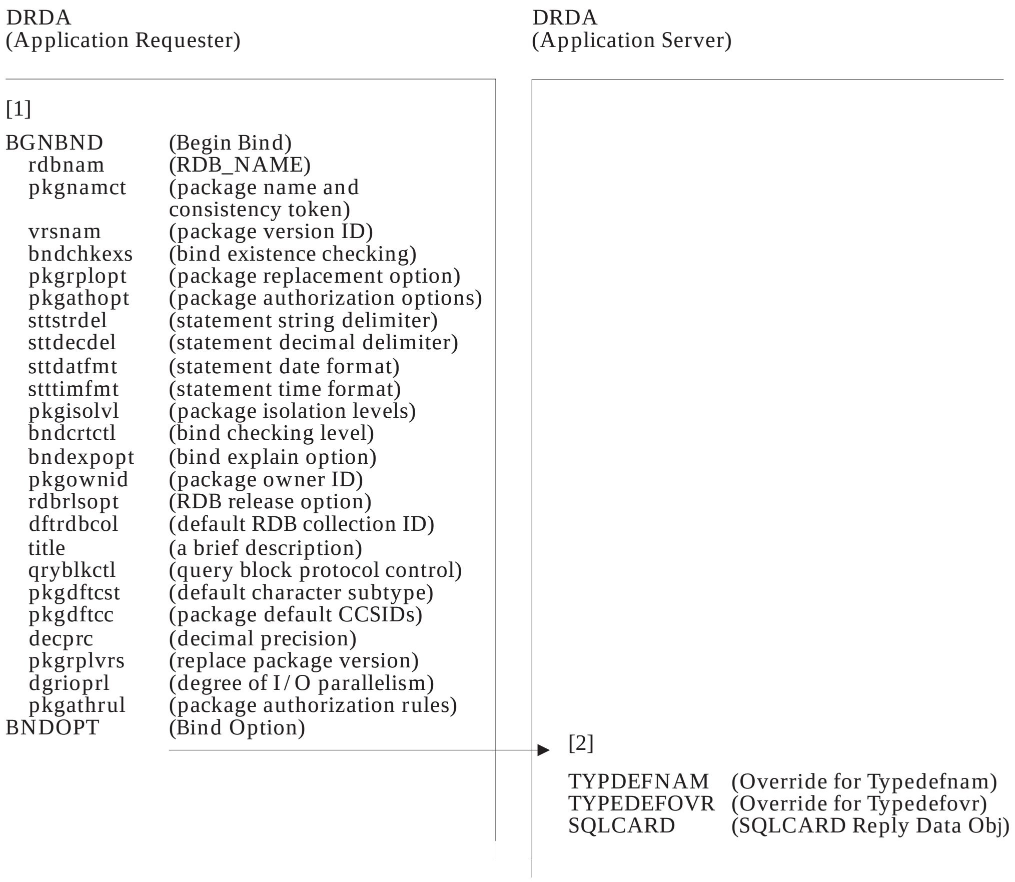
5 The application requester processes the reply. The SECCHK command is built with the SECTKN object containing the encrypted password or password substitute. For password substitution, the substitute is generated using the application requester and application server seeds. For password encryption, the received application server's connection key is used to generate the DES encryption seed k where k is equal to (Y^x mod n). The password is encrypted using the 56-bit DES encryption algorithm and the encryption seed.

6 The application server processes the SECCHK command. For password substitution, the substitute is generated using the application requester and application server seeds. If the substitute is equal to the value in the SECTKN, the user is authenticated. For password encryption, the application requester's connection key is used to generate the DES decryption seed k where k is equal to (X^y mod n). The password is decrypted using the 56-bit DES decryption algorithm and k as the seed. The userid and password is authenticated by the local security manager. The SECCHKRD is generated returning the success or failure of the authentication.

7 If the user is identified and authenticated by the application server, the RDB can be accessed.

4.4.3 Performing the Bind Operation and Creating a Package

Figure 4-5 indicates the DDM commands and replies that would flow in a normal DRDA bind processing scenario. The usual result of this process is that the application requester and the application server do the syntactic and semantic checking of the SQL statements embedded in an application and the creation of a package at the remote relational database that binds the SQL statements and host program variables in the application to the remote relational database. An application can have multiple packages on multiple application servers.



BGNBND	(Begin Bind)
rdbnam	(RDB_NAME)
pkgnamct	(package name and consistency token)
vrsnam	(package version ID)
bndchkexs	(bind existence checking)
pkgrplopt	(package replacement option)
pkgathopt	(package authorization options)
sttstrdel	(statement string delimiter)
sttdecdel	(statement decimal delimiter)
sttdatfmt	(statement date format)
stttimfmt	(statement time format)
pkgisolvl	(package isolation levels)
bndcrtctl	(bind checking level)
bndexpopt	(bind explain option)
pkgownid	(package owner ID)
rdbrlsopt	(RDB release option)
dftrdbcol	(default RDB collection ID)
title	(a brief description)
qryblkctl	(query block protocol control)
pkgdftcst	(default character subtype)
pkgdftcc	(package default CCSIDs)
decprc	(decimal precision)
pkgrplvrs	(replace package version)
dgrioprl	(degree of I/O parallelism)
pkgathrul	(package authorization rules)
BNDOPT	(Bind Option)

TYPDEFNAM	(Override for Typedefnam)
TYPEDEFOVR	(Override for Typedefovr)
SQLCARD	(SQLCARD Reply Data Obj)

Figure 4-5 DRDA Flows: Binding and/or Package Creation (Part 1)

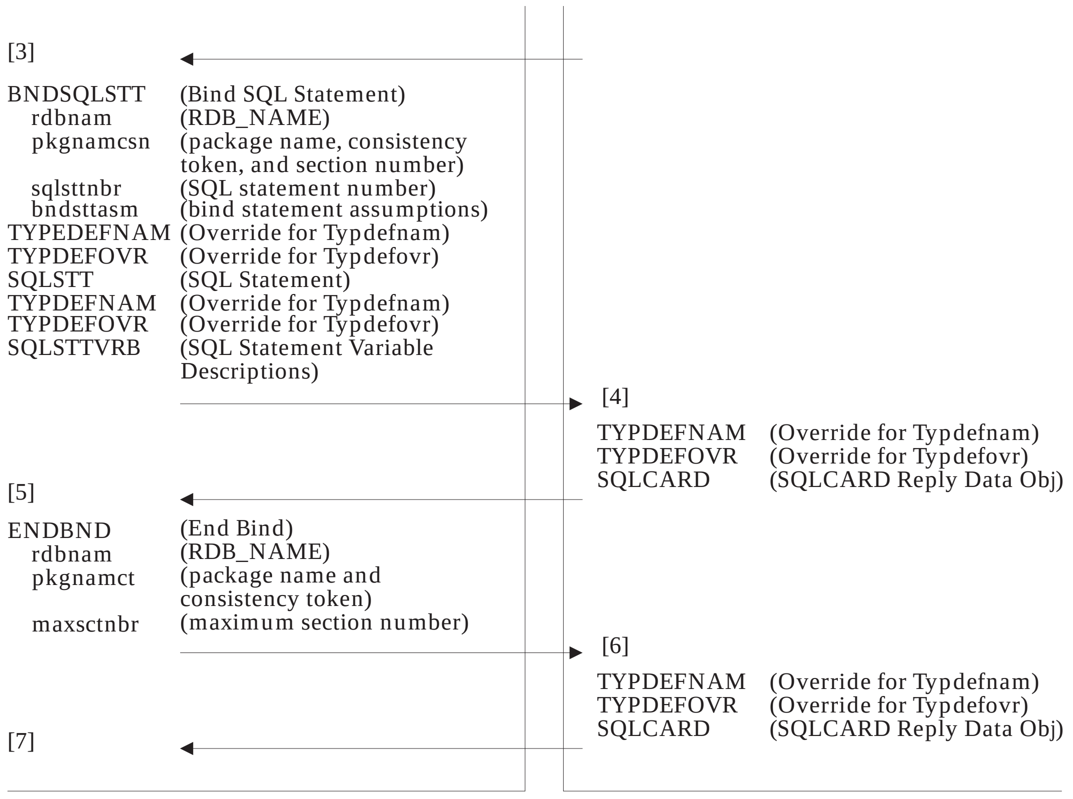

[3]
BNDSQLSTT (Bind SQL Statement)
rdbnam (RDB_NAME)
pkgnamcsn (package name, consistency token, and section number)
sqlsttnbr (SQL statement number)
bndsttasm (bind statement assumptions)
TYPEDEFNAM (Override for Typdefnam)
TYPDEFOVR (Override for Typdefovr)
SQLSTT (SQL Statement)
TYPDEFNAM (Override for Typdefnam)
TYPDEFOVR (Override for Typdefovr)
SQLSTTVRB (SQL Statement Variable Descriptions)
[4]
TYPDEFNAM (Override for Typdefnam)
TYPDEFOVR (Override for Typdefovr)
SQLCARD (SQLCARD Reply Data Obj)
[5]
ENDBND (End Bind)
rdbnam (RDB_NAME)
pkgnamct (package name and consistency token)
maxsctnbr (maximum section number)
[6]
TYPDEFNAM (Override for Typdefnam)
TYPDEFOVR (Override for Typdefovr)
SQLCARD (SQLCARD Reply Data Obj)
[7]

Figure 4-6 DRDA Flows: Binding and/or Package Creation (Part 2)

The following is a brief description of some of the parameters for the DDM commands that this document discusses. The DDM Reference provides a more detailed description of the parameters.

1 After the application requester and the application server have established the proper connection (described in Figure 4-2 on page 55), they can perform a bind operation. Other flows can precede or follow the bind flow and be part of the same unit of work.

The application requester that is acting as the agent for the application performing the bind function creates a Begin Bind (BGNBND) command, providing the name of the package and a consistency token (used to verify that the resulting package and application are synchronized during application execution) in the *pkgnamct* parameter, and desired version ID for the package in the *vrsnam* parameter. A null value in the *vrsnam* parameter indicates that no version ID is to be assigned for the package. The *pkgrplvrs* parameter can be used to specify the version name of the package to be replaced with the package being bound. The BGNBND command can also specify bind options that control various aspects of bind processing at the application server.

dgrioprl is an optional parameter[19] that informs the database to use I/O parallelism at the specified degree, if available.

19. This parameter is not supported by DRDA Remote Unit of Work (SQLAM Level 3) application requesters and application servers.

pkgathrul is an optional ignorable parameter[20] that specifies which authorization ID should be used when executing dynamic SQL in this package.

The application requester can also send additional bind options in BNDOPT command objects. This allows the application requester to send a bind option to the server for which no defined DRDA parameter or value exists.

The application requester then sends the command to the application server.

2 The application server receives the BGNBND command and determines if the package name already exists in the requested relational database. It then determines if it can support the options that the BGNBND command passes.

If the application server finds any errors in processing the BGNBND command or the bind options, it generates and returns a BGNBNDRM reply message (indicating that the Begin Bind operation failed) to the application requester.

In either case, the application server creates an SQLCARD as a reply data object and returns it to the application requester. A detailed definition of the SQLCARD is in Section 5.6.4.9 on page 190.

The optional reply data objects TYPDEFNAM and TYPDEFOVR can be supplied to override the representation specification supplied on the earlier ACCRDBRM. If specified, these reply data objects apply only until the end of the current reply; for example, only for the SQLCA being returned. See Section 5.7.1.1 on page 250 for more details.

- If the application server returns a BGNBNDRM reply message, it always precedes the SQLCARD reply data object.
- After the application server processes a BGNBND and returned a SQLCARD reply data object, which indicates that bind flows can continue for the named package, it rejects any further BGNBND commands or any other DRDA command, except BNDSQLSTT, until ENDBND,or processing that successfully ends the unit of work. See Section 4.4.12.1 on page 117 and Section 4.4.12.2 on page 120 for a description of commit and roll back processing in DRDA.

Any commands rejected for this reason receive the relational database Package Binding Process Active (PKGBPARM) reply message.

The application server also rejects any BNDSQLSTT or ENDBND commands that do not have the same value for the *pkgnamct* that appeared on the BGNBND command.

3 If the SQLCARD reply data object that the application server has returned to the application requester indicates that the BGNBND command was not successful, the application requester can change any of the parameters or options and send another BGNBND command to the application server or it can return an exception to the application that is doing the bind operation.

Assuming it receives a normal SQLCARD (no errors were indicated), the application requester continues the bind process by creating and sending Bind SQL Statement (BNDSQLSTT) commands. It creates each BNDSQLSTT command with the proper package name, consistency token, and package section number in the *pkgnamcsn*

20. This parameter is only supported by DRDA Level 1 and DRDA Level 2 application requesters and application servers at SQLAM Level 5.

parameter, the source application statement number in the *sqlsttnbr* parameter, whether there are statement assumptions in the *bndsttasm* parameter, and a single SQL statement in the SQLSTT command data object. A detailed definition of the SQLSTT is in Section 5.6.4.5 on page 185.

If the SQL statement that is being bound references any application variables, then the SQLSTTVRB command data describes these variables. If a file reference is specified as an input host variable, the application requester replaces the file reference variable in the SQLSTTVRB with the underlying base LOB SQL data type. DRDA Level 4 only supports file reference variables that the source system reads. A detailed description of the contents of the SQLSTTVRB is in Section 5.6.2.3 on page 165.

The optional command data objects TYPDEFNAM and TYPDEFOVR can be specified. When specified, they override the representation specification provided on the earlier ACCRDB command. They are effective until the end of the command or until overridden again.

See Section 5.7.1.1 on page 250 for more details.

- All SQL statements in an application program are input to the bind process at the application server with some exceptions. See rule PB9 for these exceptions. The application server determines what to do with each statement.
- The application requester must be tolerant of statements it does not understand. It cannot fail to send the SQL statement to the application server because it does not understand the syntax. The application requester must assign a unique non-zero section number to each statement it does not understand. The application server will thus be responsible for final validation of the statement. See Section 7.10 on page 296 for details.

  For all statements, the application requester must replace program variable references with the *:H* variable indicator. This is to shield the application server from program language characteristics in the host variable syntax. The description of each referenced host variable must appear in the SQLSTTVRB command data object in the exact order they are referenced in the SQL statement (the SQLSTT command data object). If the SQL statement references any host application variable more than once, it must restate that host application variable (in the proper sequence) in the SQLSTTVRB. This includes a program variable reference that specifies a procedure name within an SQL statement that invokes a stored procedure. Note, however, that the stored procedure name value flows in the *prcnam* parameter rather than in an SQLDTA on the EXCSQLSTT for that SQL statement.

  The original syntax of the referenced host variable is also included in the SQLSTTVRB description of the host variable. This information is included for diagnostic purposes.
- When both command data objects are present, the SQLSTT data object must come before the SQLSTTVRB data object.

  The application requester then sends the command to the application server.

4 The application server processes the *BNDSQLSTT* command and creates and returns an SQLCARD reply data object to the application requester. If the application server successfully processed SQLSTT, then it returns a normal SQLCARD. If the application server finds any errors, it creates and returns an SQLCARD (indicating the error) to the application requester.

5 If the SQLCARD reply data object that is returned to the application requester indicates that the BNDSQLSTT command was not successful, the application requester returns an exception to the application that is doing the bind operation.

Assuming it receives a normal SQLCARD reply data object, the application requester continues the bind process by creating and sending additional BNDSQLSTT commands to the application server.

After the application requester has processed all BNDSQLSTT commands to its satisfaction, it sends an ENDBND command to the application server.

6 The application server receives and processes the ENDBND command and creates an SQLCARD reply data object. If it finds no errors, it returns a normal SQLCARD. Otherwise, the application server indicates a single error in the SQLCARD, which is returned to the application requester. If an error results in no package at the application server, the application server must generate an SQLSTATE value that does not begin with characters 00, 01, or 02. The SQLSTATE values that begin with 00, 01, and 02 imply the package was created. All other values imply the package was not created.

7 Assuming it receives a normal SQLCARD reply data object, the application requester returns a normal indication to the application that is doing the bind operation. The application can then either start another bind operation, commit the unit of work to make the changes permanent, or roll back the unit of work to back out the changes. See Section 4.4.12.1 on page 117 and Section 4.4.12.2 on page 120 for a description of commit and rollback processing in DRDA.

If the SQLCARD reply data object that the application server has returned to the application requester indicates that the ENDBND command was not successful, the application requester returns an exception to the application that is doing the bind operation.

4.4.4 Deleting an Existing Package

Figure 4-7 indicates the DDM commands and replies that flow during a process that intends to drop an existing package from a relational database. The normal result of these flows is that the identified package no longer exists at the remote relational database, so attempts to execute statements in that package now result in error conditions.

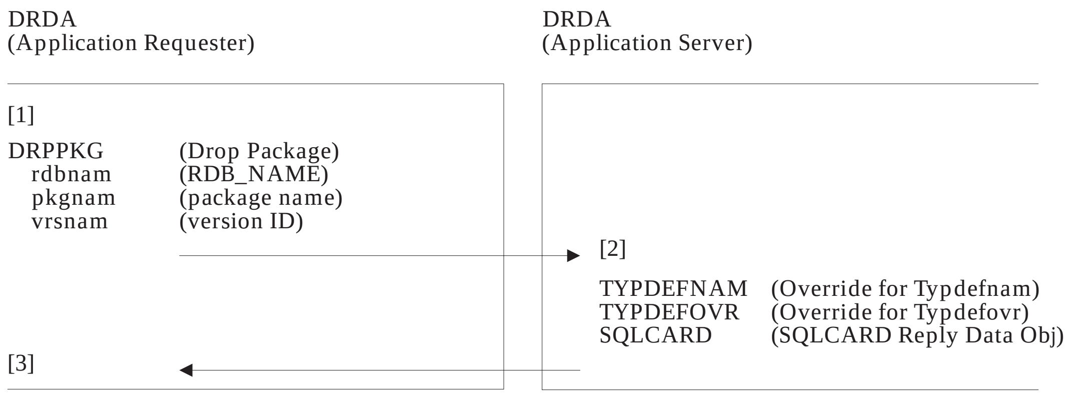


Figure 4-7 DRDA Flows: Dropping an Existing Package

The following is a brief description of some of the parameters for the DDM commands that this volume discusses. The DDM Reference provides a detailed description of the parameters.

1 An application requester can drop a package from a relational database after the application requester and application server have established the proper connection (described in Figure 4-2 on page 55). Other commands can precede or follow the drop package command and be part of the same unit of work.

The application requester that is acting as the agent for the application performing the drop package function creates the Drop Package (DRPPKG) command by providing the desired package name in the *pkgnamct* parameter and the version ID of the package in the *vrsnam* parameter. It then sends the command to the application server.

2 The application server receives and processes the DRPPKG command and creates an SQLCARD reply data object. If the version ID in the *vrsnam* parameter contains a null value, then the only version of the package identified in the *pkgnam* to be dropped is the unnamed version. If the version ID in the *vrsnam* parameter contains a non-null value, then the application server drops only that version of the package indicated in the *pkgnam* parameter.

If the application server finds no errors, it removes the package from the relational database (within the scope of the unit of work) and returns a normal SQLCARD reply data object. Otherwise, the application server indicates a single error in the SQLCARD reply data object, which is returned to the application requester, and the package remains in the relational database.

3 Assuming it receives a normal SQLCARD reply data object, the application requester returns the results to the application. The application either performs other operations within the same unit of work, which can include dropping another package, or it can commit or roll back the unit of work. See Section 4.4.12.1 on page 117 and Section 4.4.12.2 on page 120 for a description of commit and rollback processing in DRDA.

If the SQLCARD reply data object that the application server returns to the application requester indicates that the DRPPKG command was not successful, the application requester returns an exception to the application that is doing the drop operation.

4.4.5 Performing a Rebind Operation

Figure 4-8 indicates the DDM commands and replies that would flow in a normal DRDA rebind process. The usual result of this process is that the application server rebinds a previously bound package at the relational database to the same remote relational database.

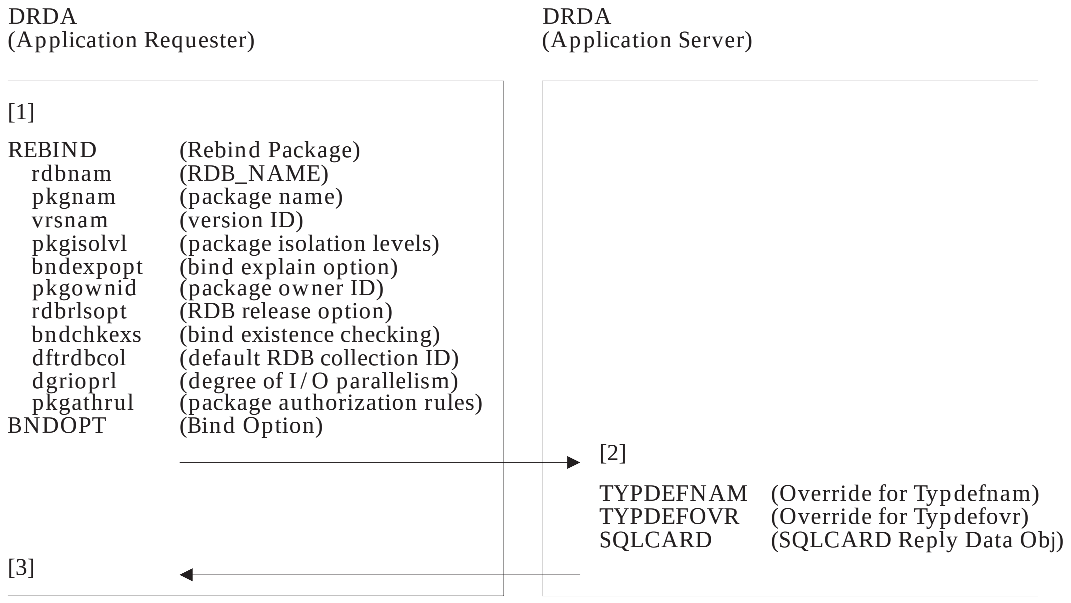

DRDA
(Application Requester)
DRDA
(Application Server)
[1]
REBIND (Rebind Package)
rdbnam (RDB_NAME)
pkgnam (package name)
vrsnam (version ID)
pkgisolvl (package isolation levels)
bndexpopt (bind explain option)
pkgownid (package owner ID)
rdbrlsopt (RDB release option)
bndchkexs (bind existence checking)
dftrdbcol (default RDB collection ID)
dgrioprl (degree of I/O parallelism)
pkgathrul (package authorization rules)
BNDOPT (Bind Option)
[2]
TYPDEFNAM (Override for Typdefnam)
TYPDEFOVR (Override for Typdefovr)
SQLCARD (SQLCARD Reply Data Obj)
[3]

Figure 4-8 DRDA Flows: Rebinding an Existing Package

The following is a discussion of the operations and functions that the application requester and the application server perform.

1 After the application requester and the application server have established the proper connection (described in Figure 4-2 on page 55), they can perform a rebind operation. Other flows can precede or follow the rebind flow and be part of the same unit of work.

The application requester that is acting as the agent for the application performing the rebind function creates a Rebind Package (REBIND) command, providing the name of the package in the *pkgnam* parameter and the version ID of the desired package in the *vrsnam* parameter.

The application requester doing the rebind also determines certain options that the application server rebinding the package needs. These include checking for the existence of all the referenced database objects and the binder's authority to access them, updating the authorizations associated with the package being replaced to show the new owner, and setting the desired isolation level that the application server will use when it executes the resulting package. These are all passed as optional parameters of the REBIND command.

dgrioprl is an optional parameter[21] that informs the database to use I/O parallelism at the specified degree, if available.

21. This parameter is not supported by DRDA Level 1 application requesters or application servers.

pkgathrul is an optional ignorable parameter[22] that specifies which authorization ID should be used when executing dynamic SQL in this package.

The application requester can also send additional bind options in BNDOPT command objects. This allows the application requester to send a bind option to the server for which no defined DRDA parameter or value exists.

The application requester then sends the command to the application server.

2 The application server receives the REBIND command and determines if the package name already exists in the requested relational database, determines if the requested version ID exists, and determines if it can support the options that the application requester has passed to it.

If the application server can support the options passed, it attempts to rebind the package to the relational database. If it successfully rebinds the package, it returns a normal SQLCA. If any errors occur, the SQLCA indicates them, and the application server cannot rebind the package. There is only one SQLCA indicating the error. SQLERRD3 contains the statement number of the first error, and SQLERRD4 contains the total number of statements with error. See Figure 5-33 on page 191 for more about SQLERRD3 and SQLERRD4.

In either case, the application server creates the SQLCA in an SQLCARD reply data object and returns it to the application requester.

3 If the SQLCARD reply data object that is returned to the application requester indicates that the REBIND command was not successful, the application requester returns an exception to the application that is doing the rebind operation.

Assuming it receives a normal SQLCARD reply data object, the application requester returns the results to the application. The application either performs other operations within the same unit of work, which can include rebinding another package, or it can commit or roll back the unit of work. See Section 4.4.12.1 on page 117 and Section 4.4.12.2 on page 120 for a description of commit and rollback processing introduced in DRDA.

22. This parameter is only supported by application requesters and application servers at SQLAM Level 5.

4.4.6 Activating and Processing Queries

Figure 4-9 on page 78 and Figure 4-11 on page 85 indicate the DDM commands and replies that occur during normal DRDA query processing. These flows produce the desired effect needed to satisfy application SQL statements of a DCL CURSOR, followed by an OPEN of the cursor, and repeated FETCHs. They also accommodate the CLOSE of a cursor before or after all the rows of the answer set have been fetched. The application requester returns the row data of the answer set (which the application server has shipped to the application requester) to the application as the application requests it.

The application server can send the row data of the answer set grouped into blocks containing varying or fixed number of rows[23] of data to the application requester per a single query request, depending on options established for the query processing. A single row of data is a special case of a fixed number of rows. For details on the description of query blocks and how they are used in the fixed row and limited row query processing protocols, see the terms QRYBLK, QRYBLKCTL, FRCFIXROW,[24] FIXROWPRC,[25] and LMTBLKPRC in the DDM Reference. Also see the rules for Section 7.19 on page 310.

If connection is between an application server and database server, any new or changed special register settings must be sent using the EXCSQLSET command prior to activating or processing queries. The EXCSQLSET command is recommended to be chained with the next SQL-related command.

Treatment of LOB Data

If any column in a query is of a LOB SQL data type, then the answer set may contain either the actual LOB data for the column or a LOB locator instantiated by the target system to represent the column in the answer set row. The application specifies how the LOB column is to be returned by the target RDB for each row retrieved in the answer set. A new parameter on the open query command allows the source system to indicate whether the form of a LOB column is specifiable with each row or only with the first row fetched in the answer set.

If a LOB locator is to be returned for a LOB column in a query, the locator value flows as locator data in the QRYDTA for the answer set.

If the LOB data value itself is to be returned for a LOB column in a query, the QRYDTA object does not contain the value bytes for the LOB data column. Instead each such LOB data column in the answer set is represented by an FD:OCA placeholder that contains only the length of the LOB data. The value bytes in the data column are externalized and returned in an object called the EXTDTA object. For more information about the FD:OCA placeholder, refer to Section 5.5.3 on page 157.

Each answer set row is returned in one or more query blocks. Each such row is called a base row; it contains every column in the answer set in the form of either the data for that column or an FD:OCA placeholder for the column if the data is externalized. If a base row has FD:OCA placeholders for externalized FD:OCA data, the corresponding EXTDTAs are said to be associated with the base row. Likewise, the EXTDTAs corresponding to the FD:OCA placeholders in a QRYDTA object are said to be associated with the QRYDTA object.

23. A block containing a fixed number of rows is limited to a single row in DRDA Remote Unit of Work.

24. Term defined as FRCSNGROW in DDM Level 3 documentation.

25. Term defined as SNGROWPRC in DDM Level 3 documentation.

LOB data may also be sent as input host variables for a query. The non-LOB columns flow as base input data in an SQLDTA command data object. LOB columns that are sent as LOB locators flow in the SQLDTA as locator data, while LOB columns sent as data are represented by FD:OCA placeholders in the SQLDTA. The data for LOB columns flows in EXTDTAs that follow the SQLDTA. Such EXTDTAs are said to be associated with the base input data.

Refer to the following DDM terms for a description of query blocks and EXTDTAs objects and how they are used in the fixed row and limited row query processing protocols:

- FIXROWPRC
- LMTBLKPRC
- EXTDTA
- EXTDTAOVR
- QRYBLK

The following sections describe the various flows that show how the application server returns row data of the answer set to the application requester and how the application requester requests more row data of the answer set from the application server, if it is available.

4.4.6.1 Fixed-Row Protocol

Figure 4-9 indicates the DDM commands used in the fixed row protocol query processing flows.

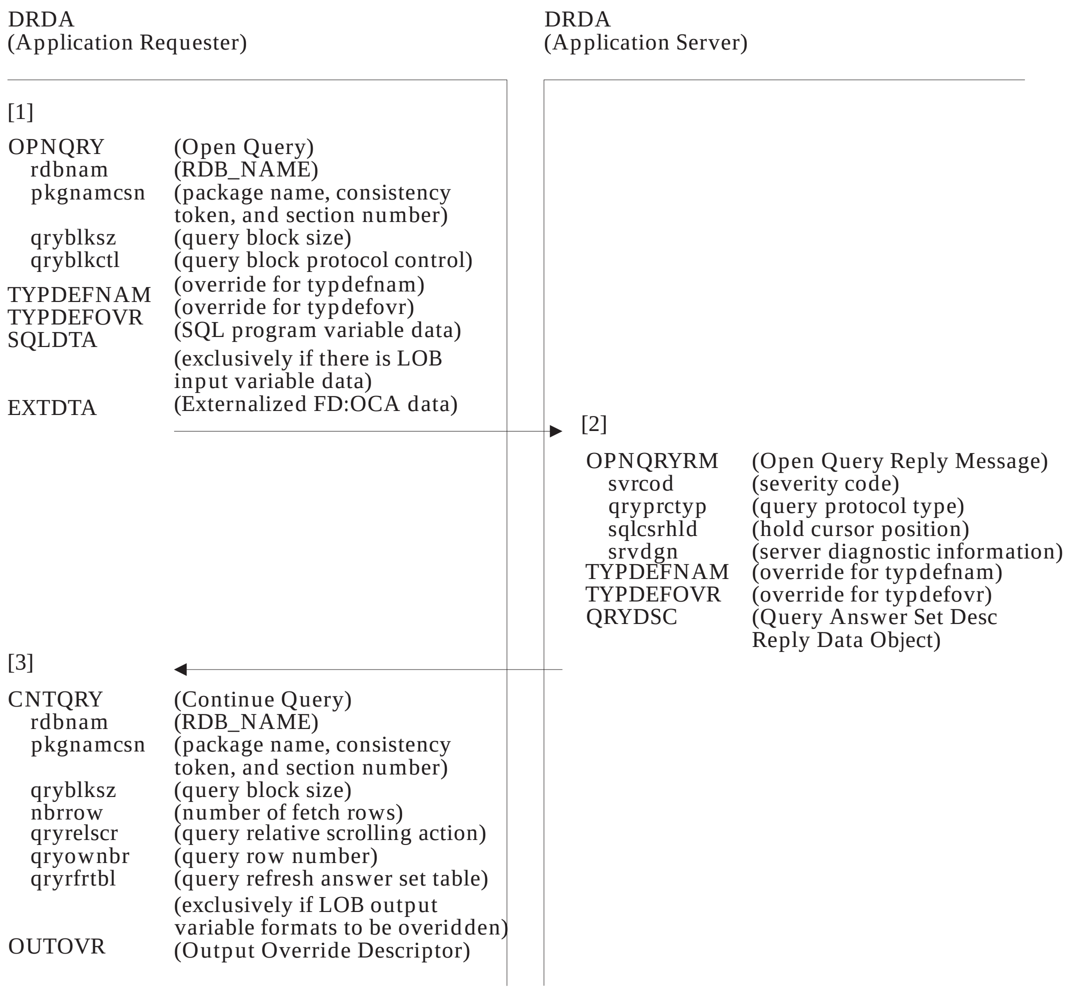

DRDA
(Application Requester)
DRDA
(Application Server)
[1]
OPNQRY (Open Query)
rdbnam (RDB_NAME)
pkgnamcsn (package name, consistency token, and section number)
qryblksz (query block size)
qryblkctl (query block protocol control)
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLDTA (SQL program variable data)
(exclusively if there is LOB input variable data)
EXTDTA (Externalized FD:OCA data)
[2]
OPNQRYRM (Open Query Reply Message)
svrcod (severity code)
qryprctyp (query protocol type)
sqlcsrhld (hold cursor position)
srvdgn (server diagnostic information)
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
QRYDSC (Query Answer Set Desc Reply Data Object)
[3]
CNTQRY (Continue Query)
rdbnam (RDB_NAME)
pkgnamcsn (package name, consistency token, and section number)
qryblksz (query block size)
nbrrow (number of fetch rows)
qryrelscr (query relative scrolling action)
qryownbr (query row number)
qryrfrtbl (query refresh answer set table)
(exclusively if LOB output variable formats to be overidden)
OUTOVR (Output Override Descriptor)

Figure 4-9 DRDA Flows: Fixed-Row Protocol Query Processing (Part 1)

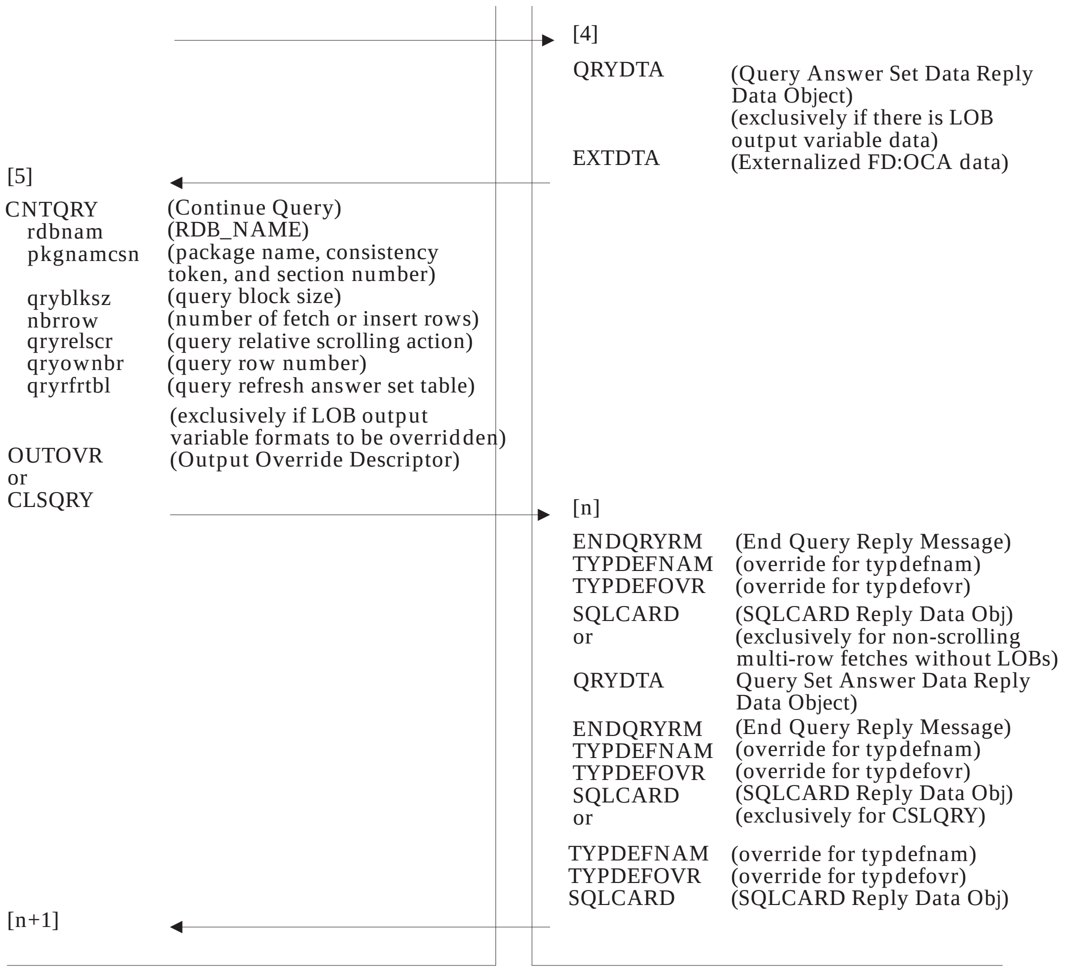

[4]
QRYDTA (Query Answer Set Data Reply Data Object)
(exclusively if there is LOB output variable data)
EXTDTA (Externalized FD:OCA data)
[5]
CNTQRY (Continue Query)
rdbnam (RDB_NAME)
pkgnamcsn (package name, consistency token, and section number)
qryblksz (query block size)
nbrrow (number of fetch or insert rows)
qryrelscr (query relative scrolling action)
qryownbr (query row number)
qryrfrtbl (query refresh answer set table)
(exclusively if LOB output variable formats to be overridden)
OUTOVR (Output Override Descriptor)
or
CLSQRY
[n]
ENDQRYRM (End Query Reply Message)
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLCARD (SQLCARD Reply Data Obj)
or (exclusively for non-scrolling multi-row fetches without LOBs)
QRYDTA Query Set Answer Data Reply Data Object)
ENDQRYRM (End Query Reply Message)
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLCARD (SQLCARD Reply Data Obj)
or (exclusively for CSLQRY)
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLCARD (SQLCARD Reply Data Obj)
[n+1]

Figure 4-10 DRDA Flows: Fixed-Row Protocol Query Processing (Part 2)

The following is a discussion of the operations and functions that the application requester and the application server performs. Here is a brief description of some of the parameters for the DDM commands. The DDM Reference provides a detailed description of the parameters.

1 After the application requester and the application server establish the proper connection (described in Figure 4-2 on page 55), an application can do an OPEN CURSOR request to the application requester. The application requester that is acting as the agent for the application performing the open cursor function creates an Open Query (OPNQRY) command providing the proper package name, consistency token, and section number in the *pkgnamcsn* parameter. It also provides the query block size (the size of the query blocks that the application server can return) that it desires in the *qryblksz* parameter. The *qryblkctl* parameter specifies whether fixed-row query protocols[26] must be forced on the opened database cursor. If the query being opened does not contain this parameter, then the application server selects the query protocol to be used as specified in the package (see *qryblkctl* in Figure 4-5 on page 67). The application requester places any input variables from the application in the SQLDTA command data object and sends the command and data to the application server. For

each of the input variables that are of a LOB SQL data type, the following occurs:

- If a LOB locator, the locator is sent with the SQLDTA object as a LOB locator DRDA type.
- If LOB data, the application requester creates an FD:OCA placeholder for the data in the SQLDTA and creates an EXTDTA object to contain the LOB value bytes obtained from memory.
- If LOB file reference variable, the application requester creates an FD:OCA placeholder for the data in the SQLDTA and creates an EXTDTA object to contain the LOB value bytes obtained from the referenced file.

The application requester sends the OPNQRY command, the SQLDTA object, and the associated EXTDTA objects, to the application server in the order listed, with EXTDTAs flowing in the same order that their corresponding host variables were specified by the application.

If the application data is not in the representation declared at ACCRDB, then the optional objects TYPDEFNAM and TYPDEFOVR must be supplied. This will allow the application server to correctly interpret the data that the application supplied. This override applies to all the data that follows the override specification until either another override is encountered or until the end of the command is reached. It is effective for data flowing from the application requester to the application server.

Note: The block size specified in the *qryblksz* must be equal to or greater than 512 bytes and equal to or less than 32,767 bytes. If not, the application server returns the VALNSPRM reply message and the application server does not execute the command.

2 The application server receives and processes the OPNQRY command. The application server determines that a fixed number of rows are to be returned per request because the application can make multi-row fetches,[27] or can update the rows or delete the rows through the cursor instance, or the cursor might have been declared scrollable.[28] The application requester might not have been aware of the update capability of this cursor instance.

If the application server successfully opens the indicated cursor, it creates an OPNQRYRM reply message. If there is a warning SQLCA returned from the relational database, an SQLCARD reply object will be built and will follow the OPNQRYRM. If the query returns any LOB columns, then the application server must select the FIXROWPRC if the application server indicates that output overrides may be sent with each CNTQRY command. The application server also generates an FD:OCA data descriptor of the row data in QRYDSC reply data object that follows either OPNQRYRM or OPNQRYRM / SQLCARD reply sequence.

The application server then generates an FD:OCA data descriptor that describes each returned row. The application server places the FD:OCA data descriptor of the row data in the QRYDSC reply data object and sends it to the application requester. Section 5.5.3.1 on page 159 gives a detailed definition of the QRYDSC.

26. Also known as single row query protocols.

27. Multi-row fetches are not supported in DRDA Remote Unit of Work, at SQLAM Level 3.

28. Scrollable cursors are not supported in DRDA Remote Unit of Work, at SQLAM Level 3.

If the data retrieved from the relational database is not in the representation declared at ACCRDBRM time, then the optional reply data objects TYPDEFNAM and TYPDEFOVR must be supplied. These reply data objects will allow the application requester to correctly interpret the data that the database management system supplied. This override applies to all the data that follows the override specification. For user data defined by this command, the overrides stay in effect until the data is exhausted or the cursor is closed. The override remains in effect for any user data returned by CNTQRY commands. The override does not apply to an SQLCARD following an ENDQRYRM sent in response to CNTQRY. This override is in effect for data flowing from the application server to the application requester.

- The application server sends the QRYDSC reply data object. The QRYDSC must follow the OPNQRYRM reply message and if present, a warning SQLCARD.
- QRYDSC contains the description of an SQLCA and the row data. The application server sends the SQLCA with each row of data in the QRYDTA reply data object. This indicates any condition that can be present as a result of the row retrieval. See Section 5.3 on page 145for detail on QRYDSC.
- If the OPNQRYRM reply message and the QRYDSC reply data object cannot be contained within a single query block of the indicated size, then the OPNQRYRM reply message appears first in the first query block and the QRYDSC is placed immediately after it and can flow across as many additional query blocks as is required to hold it. If the last block is not full, it is truncated at the end of the data descriptor.
- The application server actually sends a QRYDSC reply data object that flows across multiple query blocks as multiple QRYDSC reply data objects. Each QRYDSC reply data object takes up a full query block except the first, and potentially the last, one. The application requester pulls the multiple reply data objects back together into a single QRYDSC data object.
- In response to an OPNQRY command, if the application server is not going to send the QRYDSC reply data object, then a DDM error reply message must appear first in the query block. For those reply messages that require an SQLCARD, the SQLCARD reply data object, indicating the condition, follows the reply message in the query block that is sent.
- In response to an OPNQRY command for a query that is currently suspended (previously opened and has not been terminated), the application server returns a QRYPOPRM reply message first in the query block.

3 The application requester receives the OPNQRYRM or OPNQRYRM/SQLCARD reply message and QRYDSC reply data object from the application server and indicates to the application that open processing was successful.

If the application specified an SQLDA to be used for the FETCH, then the application requester may create an OUTOVR object to send to the application server as command data for the CNTQRY command. The OUTOVR object is not required if there are no LOB data columns in the row, but may be optionally sent by the application requester.[29] If multi-row fetch is requested, then all rows are fetched using the same SQLDA.

If any LOB data columns are in the row and the application wishes to receive a LOB column in a form other than as data, then the application requester must create an OUTOVR to send to the application server. The application server uses the overriding OUTOVR to determine the format of the data returned. If no OUTOVR is sent with the CNTQRY command, the application server fetches the data using the last OUTOVR sent with a CNTQRY command for this query, or if none has been sent, it uses the QRYDSC returned with the OPNQRYRM.

Note: At any time after the application requester has sent the OPNQRY command and the application server has successfully processed it, and before the application server has sent an ENDQRYRM reply message, the application requester can send a Close Query (CLSQRY) command with the correct package name, consistency token, and section number in the *pkgnamcsn* parameter to the application server.

If the application server processes the CLSQRY command successfully, it terminates the query and sends the application requester an SQLCARD reply data object indicating that it has closed the query.

If it has already terminated (or not opened) the query, the application server sends a QRYNOPRM reply message to the application requester indicating the query is not open.

If the application requester receives a reply message indicating an error occurred, it notifies the application of an error condition.

When the application requests the first row or rows (if multi-row fetches) from the application requester, the application requester creates a Continue Query (CNTQRY) command with the same value for *pkgnamcsn* that it supplied on the corresponding OPNQRY command and sends CNTQRY and the optional OUTOVR object to the application server. The application requester reflects the application requested multi-row fetch and scrolling options in the *nbrrow*, *qryrelscr*, *qryrownbr*, and *qryrfrtbl* parameters.[30] The application requester can supply a different value in the *qryblksz* parameter.

4 The application server receives the CNTQRY command and the OUTOVR, if present. It processes the OUTOVR to create an SQLDA for passing to the relational database for use when fetching the row or rows. It retrieves the first row or rows of the answer set, places it in a QRYDTA reply data object with an SQLCA preceding each row, and sends it to the application requester. The row or rows returned depend on the multi-row fetch and scrolling parameters sent on CNTQRY, as well as the presence of LOB data in the row.

If any LOB columns are in the row being retrieved, data will be returned as follows: All LOB locators are returned in the QRYDTA object. Each LOB column is represented by an FD:OCA placeholder in the QRYDTA and the LOB value bytes are returned in an EXTDTA object following the QRYDTA. If more than one EXTDTA is returned, they are returned in the order that their corresponding FD:OCA placeholders occur in the QRYDTA object.

29. This is true for all SQLTYPEs that require server resolution of compatible data types. LOG SQLTYPEs are the only such at this time.

30. These parameters are not supported in DRDA Level 1.

See Section 5.5.3.1 on page 159 for a detailed definition of QRYDTA.

- If a single row of the answer set data or the answer set of a multi-row fetch (contained in a QRYDTA reply data object) cannot be contained within a single query block of that size, then it will span two or more query blocks. If the last block is not full, it is truncated at the end of the data.
- A QRYDTA data object that flows across multiple query blocks is actually sent as multiple QRYDTA data objects. Each object takes up a full query block except (potentially) the last one. The application requester must pull the reply data objects back together into a single QRYDTA data object.
- In response to a CNTQRY command, if the application server is not going to send the QRYDTA reply data object, then a DDM error reply message must appear first in the query block. For those reply messages that require an SQLCARD, the SQLCARD reply data object, indicating the condition, follows the reply message in the query block that is sent.
- The response to the previous OPNQRY command defined the data that flows to the application for this command. No further representation overrides are allowed for row data.

5 The application requester receives the QRYDTA reply data object and maps the row data to the application's host variables. The application requester obtains LOB column value bytes from the associated EXTDTA objects that follow the QRYDTA.

When the application requests the next row or rows from the application requester, the application requester creates another CNTQRY command with the same value for *pkgnamcsn* that it supplied on the corresponding OPNQRY command. The application requester reflects application requested multi-row fetch and scrolling options in the *nbrrow*, *qryrelscr*, *qryrownbr*, and *qryrfrtbl* parameters. The application requester can supply a different value in the *qryblksz* parameter. It then sends CNTQRY to the application server.

Steps 4 and 5 are repeated until the application does not request any more rows, the application closes the cursor, or in the case of non-scrolling cursors, there are no more rows of the answer set available.

Note: For multi-row fetches, the application requester must provide a statement level SQLCA to the application. See **Building the Statement-Level SQLCAs for Multi-Row Fetch Operations** on page 437 for guidance in building the statement level SQLCA.

n For non-scrolling cursors, or queries (except dynamic queries with HOLD), if the application server receives a CNTQRY command and fewer rows than requested are in the answer set (this can even occur on the first CNTQRY command), the application server generates an ENDQRYRM reply message and sends it to the application requester followed by an SQLCARD reply data object that indicates the end of query processing condition (SQLSTATE=02000). For multi-row fetches on a non-scrolling cursor, there can be some rows returned before returning the ENDQRYRM. The application server then closes the cursor.

For cursors that scroll or dynamic queries with the HOLD option, a CNTQRY that runs out of rows in the answer set does not result in an ENDQRYRM and closed cursor. The condition is reflected in the SQLCARD returned for the CNTQRY and if the cursor is scrollable, the application can reposition the cursor for future fetches, or the application can close the cursor.

- At any time during query processing, the relational database might incur a problem that causes the query to be terminated. For non-scrolling cursors, the application server sends the ENDQRYRM reply message, followed by an SQLCARD reply data object, which indicates the reason for failing to return another data row of the answer set. If the error occurs during a multi-row fetch, the good rows are returned with the ENDQRYRM and SQLCARD with the error indication.

  For cursors that scroll, an ENDQRYRM is not returned. The error condition is returned in the SQLCARD on the next CNTQRY.

- TYPDEFNAM and TYPDEFOVR can be sent before the SQLCARD to override the descriptions. Any TYPDEFNAM or TYPDEFOVR sent in response to the OPNQRY or a previous CNTQRY does not affect the description of the SQLCARD.

n+1 For non-scrolling cursors, when the application requester receives the ENDQRYRM reply message, it knows that it has received the last row of answer data and that the application server has closed the query, so it does not send any additional CNTQRY commands to the application server.

The application requester receives an SQLCARD reply data object and reports the indicated condition to the application.

If the application requester receives a request to CLOSE the cursor at this point, it does not need to communicate with the application server as it knows the cursor is already closed.

At this point, the application/application requester can continue with additional defined DRDA flows with the resulting changes being in the same unit of work or it can complete the unit of work in some defined fashion.

Note: The execution of a ROLLBACK, through any method, causes the termination of a query. The execution of a COMMIT, through any method, causes the termination of a query, except for queries with the HOLD option in the DECLARE CURSOR statement.

4.4.6.2 Limited Block Protocol (No LOB Data in Answer Set)

Figure 4-11 indicates the DDM commands used by the limited block query processing flows when there are no LOB outputs. Refer to Section 4.4.6.3 on page 90 if there are LOB data columns in the answer set.

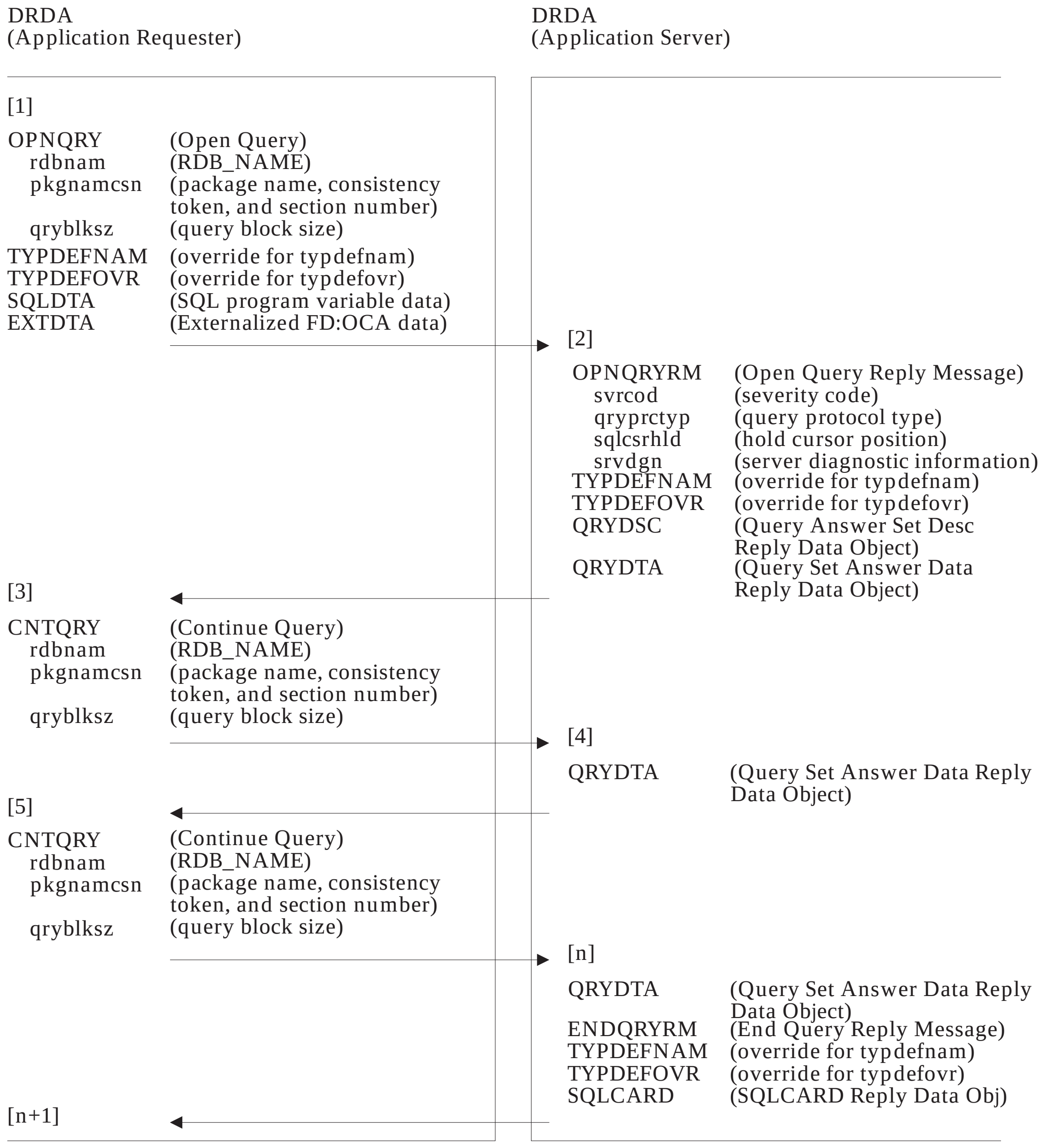

DRDA
(Application Requester)
DRDA
(Application Server)
[1]
OPNQRY (Open Query)
rdbnam (RDB_NAME)
pkgnamcsn (package name, consistency token, and section number)
qryblksz (query block size)
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLDTA (SQL program variable data)
EXTDTA (Externalized FD:OCA data)
[2]
OPNQRYRM (Open Query Reply Message)
svrcod (severity code)
qryprctyp (query protocol type)
sqlcsrhld (hold cursor position)
srvdgn (server diagnostic information)
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
QRYDSC (Query Answer Set Desc Reply Data Object)
QRYDTA (Query Set Answer Data Reply Data Object)
[3]
CNTQRY (Continue Query)
rdbnam (RDB_NAME)
pkgnamcsn (package name, consistency token, and section number)
qryblksz (query block size)
[4]
QRYDTA (Query Set Answer Data Reply Data Object)
[5]
CNTQRY (Continue Query)
rdbnam (RDB_NAME)
pkgnamcsn (package name, consistency token, and section number)
qryblksz (query block size)
[n]
QRYDTA (Query Set Answer Data Reply Data Object)
ENDQRYRM (End Query Reply Message)
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLCARD (SQLCARD Reply Data Obj)
[n+1]

Figure 4-11 DRDA Flows: Limited Block Protocol Query Processing (No LOB Data)

The following is a discussion of the operations and functions the application requester and the application server perform. This is just a brief description of some of the parameters for the DDM commands. See the DDM Reference for a detailed description of the parameters.

1 After the application requester and the application server have established the proper connection (described in Figure 4-2 on page 55), an application can send an OPEN CURSOR request to the application requester. The application requester acting as the agent for the application performing the open cursor function, creates an Open Query

(OPNQRY) command providing the proper package name, consistency token, and section number in the *pkgnamcsn* parameter. It also provides the desired query block size (the size of the query blocks that the application server can return) in the *qryblksz* parameter.

The *qryblkctl* parameter specifies whether fixed row protocols must be forced on the opened database cursor. If the query being opened does not include this parameter, then the application server selects the query protocol to be used based on information in the package (see *qryblkctl* in Figure 4-5 on page 67). Its absence here allows limited block processing. The application requester places any input variables from the application in the SQLDTA command data object and sends the command and the data to the application server. Input host variables containing LOB data types are handled as in Section 4.4.6.1 on page 78.

Note: The block size specified in the *qryblksz* must be equal to or greater than 512 bytes and equal to or less than 32,767 bytes. If not, the application server returns the VALNSPRM reply message and the application server does not execute the command.

2 The application server receives and processes the OPNQRY command. It then determines that it will use limited block protocols because no SQL UPDATEs or DELETEs are to be performed against the corresponding cursor, and the OPNQRY command did not include the *qryblkctl* parameter. The *qryblksz*, which the application requester has established and sent on the OPNQRY command, determines the size of each query block.

If the application server successfully opens the cursor, it creates and sends an OPNQRYRM reply message to the application requester.

If the relational database returned a warning SQLCA, then an SQLCARD will be sent.

The application server then generates an FD:OCA data descriptor that describes each returned row. This description is placed in the QRYDSC reply data object that in turn is placed in the first query block after the OPNQRYRM reply message or warning SQLCARD, which the application server will return to the application requester.

The application server can also create a QRYDTA reply data object and place all or part of the first data row and all or part of any of the next data rows in it until the QRYDTA fills the rest of the last query block containing the last QRYDSC reply data object.

- The application server sends the QRYDSC reply data object. The QRYDSC must follow the OPNQRYRM reply message or warning SQLCARD.
- QRYDSC contains the description of an SQLCA and the row data. The application server sends the SQLCA with each row of data in the QRYDTA reply data object. This indicates any condition that can be present as a result of the row retrieval. See Section 5.3 on page 145 for detail on QRYDSC.
- If the OPNQRYRM reply message and the QRYDSC reply data object cannot be contained within a single query block of the indicated size, then the OPNQRYRM reply message and optional SQLCARD appear first in the first query block and the QRYDSC is placed immediately after them. They can flow across as many additional query blocks as is required to hold them.
- A QRYDSC reply data object that flows across multiple query blocks is actually sent as multiple QRYDSC reply data objects. Each QRYDSC reply data object takes up a full query block except the first and (potentially) the last one. The application requester must pull the multiple reply data objects back together into a single

QRYDSC data object.

- If space is available in the last query block to be sent, the application server can create a QRYDTA reply data object to fill the space that is left in this query block (which contains some or all of the QRYDSC reply data object). The QRYDTA reply data object can contain a single row, multiple rows, and/or a partial data row of the answer set.
- For queries (except dynamic queries with HOLD), if the application server can place all of the data rows in the answer set in the last query block and leave enough space for the ENDQRYRM reply message and an SQLCARD reply data object, then it places an ENDQRYRM reply message and the SQLCARD reply data object in that query block and returns them to the application requester. If the ENDQRYRM reply message and the SQLCARD reply data object do not fit in the remaining space, then the application server does not send either to the application requester in this query block.

  If the application server sends the ENDQRYRM reply message and the SQLCARD reply data object to the application requester, then the application server terminates the query.
- In response to an OPNQRY command, if the application server is not going to send the QRYDSC reply data object, then a DDM error reply message must appear first in the first query block.
- In response to an OPNQRY command for a query that is currently suspended (previously opened and has not been terminated), the application server returns a QRYPOPRM reply message first in the query block.

3 The application requester receives the OPNQRYRM reply message and QRYDSC reply data object from the application server and indicates to the application that open processing was successful.

Note: At any time after the application requester has sent the OPNQRY command and the application server has successfully processed it, and before the application server has sent an ENDQRYRM reply message, the application requester can send a Close Query(CLSQRY) command with the correct package name, consistency token, and section number in the *pkgnamcsn* parameter to the application server.

If the application server processes the CLSQRY command successfully, it terminates the query and sends the application requester an SQLCARD reply data object indicating the application server has closed the query.

If the application server had already terminated (or not opened) the query, then it sends a QRYNOPRM reply message to the application requester indicating the query is not open.

The application requester receives the reply data object from the application server and processes it. If the application requester receives QRYDSC and QRYDTA reply data objects, the first row data received in the QRYDTA reply data object is mapped to the application's host variables (when the application performs the first FETCH operation) based on the description passed in the QRYDSC reply data object.

When the application requests the next row from the application requester (performs the next FETCH operation), the application requester maps the next row from the QRYDTA reply data object to the application's host variables.

When the application performs a FETCH and a complete row is no longer available in the QRYDTA reply data object, the application requester creates a CNTQRY command with the same value for *pkgnamcsn* as the corresponding OPNQRY command supplied. A different value can be supplied in the *qryblksz* parameter and will result in the application server returning a new size for the next query block.

4 When the application server receives the CNTQRY command, if it has a partial row from the last OPNQRY or CNTQRY command, it places the partial row in the QRYDTA reply data object. If it does not have a partial row, then it retrieves the next data row from the answer set and places it in the QRYDTA reply data object with an SQLCA. The first row or partial row returned can span query blocks. The application server can continue to retrieve rows and place them in the QRYDTA reply data object until the query block is full.

- The application server actually sends a QRYDTA data object that flows across multiple query blocks as multiple QRYDTA data objects. Each object takes up a full query block except (potentially) the last one created for a particular query. Then the application requester must pull the reply data objects back together into a single QRYDTA data object.
- For queries (except dynamic queries with HOLD), if the application server can place all of the data rows in the answer set in the last query block and leave enough space in that query block for the ENDQRYRM reply message and an SQLCARD reply data object, then the application server places an ENDQRYRM reply message and the SQLCARD reply data object in that query block and returns them to the application requester. If the ENDQRYRM reply message and the SQLCARD reply data object do not fit in the space remaining, then the application server will not send either to the application requester in this query block. They will be sent as the only responses to the next CNTQRY command.

  If the application server sends the ENDQRYRM reply message and the SQLCARD reply data object to the application requester, then the query is terminated.
- In response to a CNTQRY command, if the application server is not going to send the QRYDTA reply data object, then a DDM error reply message must appear first in the query block. For those reply messages that require an SQLCARD, the SQLCARD reply data object, indicating the condition, follows the reply message in the query block that is sent.

The application server then sends the query block to the application requester.

5 When the application requester receives the QRYDTA reply data object, it passes the row data that spanned the previous query block and the one just received (if any) to the application. The application requester maps the row data it received in the QRYDTA reply data object to the application's host variables (in response to the previously unsatisfied FETCH operation) based on the description passed in the QRYDSC reply data object at the beginning of the query processing.

When the application requests the next row from the application requester, the application requester maps the next row from the QRYDTA reply data object to the application's host variables.

When a complete row is no longer available in the query block, the application requester creates a CNTQRY command with the same value for *pkgnamcsn* as the corresponding OPNQRY command supplied. The *qryblksz* parameter can contain a different value and will result in a new size for the next query block the application server will return.

Steps 4 and 5 are repeated until the application server returns the QRYDTA reply data object with all of the last row of the answer set to the application requester, or until the application does not request any more rows.

n When the application server receives the CNTQRY command, if it has a partial row from the last OPNQRY or CNTQRY command it places the partial row in the QRYDTA reply data object. If it does not have a partial row, then it retrieves the next data row from the answer set and places it in a QRYDTA reply data object with an SQLCA. The first row or partial row returned can span query blocks. The application server continues to retrieve additional rows of the answer set and to place them in the QRYDTA reply data object until it has retrieved the last row.

For queries (except dynamic queries with HOLD), after it has placed the last row of the answer set (which completes the last QRYDTA reply data object) in the query block, the application server generates an ENDQRYRM reply message and an SQLCARD reply data object and places them in the query block.

The application server then closes the query and will no longer accept CNTQRY commands for this cursor until an OPNQRY is again processed for this cursor.

The application server then sends the last query block to the application requester.

n+1 When the application requester receives the reply block, it passes the row data that spanned the previous QRYDTA reply data object and the one just received (if any) to the application. The application requester maps the row data it received in the QRYDTA reply data object to the application's host variables (in response to the previously unsatisfied FETCH operation) based on the description passed in the QRYDSC reply data object at the beginning of the query processing.

When the application requests the next row from the application requester, the application requester maps the next row from the QRYDTA reply data object to the application's host variables.

When the application requester receives the ENDQRYRM reply message, it knows that the application server has processed the last row of answer data. It knows the application server has closed the query, so the application requester will not send any additional CNTQRY commands to the application server.

If the application requester receives a request to CLOSE the cursor at this point, it does not need to communicate with the application server as it knows the cursor is already closed.

At this point, the application or the application requester can continue with additional defined DRDA flows with the resulting changes being in the same unit of work, or it can complete the unit of work in some defined fashion.

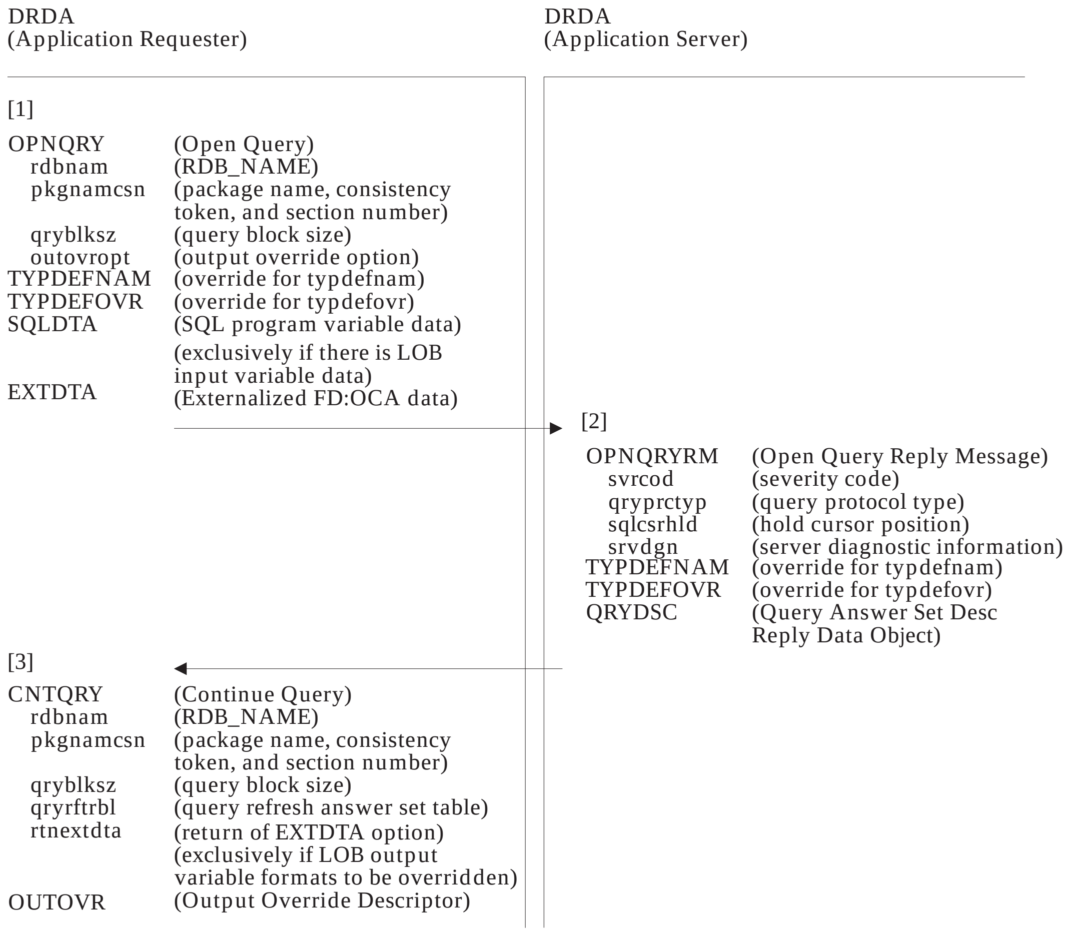
Note: The execution of a ROLLBACK, through any method, causes the termination of a query. The execution of a COMMIT, through any method, causes the termination of a query, except for queries with the HOLD option in the DECLARE CURSOR statement.

4.4.6.3 Limited Block Protocol (LOB Data in Answer Set)

The *rtnextdta* parameter determines the major processing flows when there are LOB data columns in the answer set.

Figure 4-12 and Figure 4-13 on page 91 indicate the complete limited block processing flows when the *rtnextdta* value is *rtnextall*.

Figure 4-12 and Figure 4-13 on page 91 also indicate partial limited block processing flows when the *rtnextdta* value is *rtnextrow*. In addition, refer to Figure 4-14 on page 94 for additional commands that apply in this case.



```
DRDA                                              DRDA
(Application Requester)                           (Application Server)

[1]

OPNQRY         (Open Query)
   rdbnam      (RDB_NAME)
   pkgnamcsn   (package name, consistency
               token, and section number)
   qryblksz    (query block size)
   outovropt   (output override option)
TYPDEFNAM      (override for typdefnam)
TYPDEFOVR      (override for typdefovr)
SQLDTA         (SQL program variable data)
               (exclusively if there is LOB
               input variable data)
EXTDTA         (Externalized FD:OCA data)
               ------------------------------->  [2]

                                                   OPNQRYRM     (Open Query Reply Message)
                                                      svrcod    (severity code)
                                                      qryprctyp (query protocol type)
                                                      sqlcsrhld (hold cursor position)
                                                      srvdgn    (server diagnostic information)
                                                   TYPDEFNAM    (override for typdefnam)
                                                   TYPDEFOVR    (override for typdefovr)
                                                   QRYDSC       (Query Answer Set Desc
                                                                Reply Data Object)
[3]            <-------------------------------
CNTQRY         (Continue Query)
   rdbnam      (RDB_NAME)
   pkgnamcsn   (package name, consistency
               token, and section number)
   qryblksz    (query block size)
   qryrftrbl   (query refresh answer set table)
   rtnextdta   (return of EXTDTA option)
               (exclusively if LOB output
               variable formats to be overridden)
OUTOVR         (Output Override Descriptor)
```

Figure 4-12 Limited Block Protocol (with LOB Data, rtnextall) (Part 1)

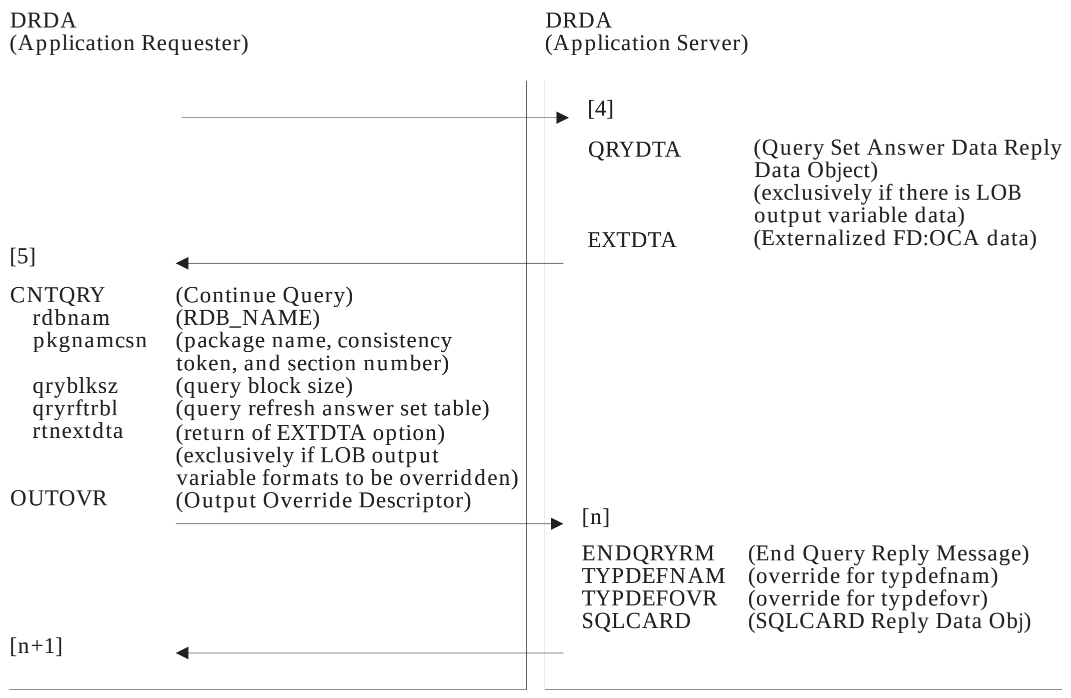

DRDA
(Application Requester)
DRDA
(Application Server)
[4]
QRYDTA (Query Set Answer Data Reply Data Object)
(exclusively if there is LOB output variable data)
EXTDTA (Externalized FD:OCA data)
[5]
CNTQRY (Continue Query)
rdbnam (RDB_NAME)
pkgnamcsn (package name, consistency token, and section number)
qryblksz (query block size)
qryrftrbl (query refresh answer set table)
rtnextdta (return of EXTDTA option)
(exclusively if LOB output variable formats to be overridden)
OUTOVR (Output Override Descriptor)
[n]
ENDQRYRM (End Query Reply Message)
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLCARD (SQLCARD Reply Data Obj)
[n+1]

Figure 4-13 Limited Block Protocol (with LOB Data, rtnextall) (Part 2)

Limited Block Protocol (rtnextdta=rtnextall)

This discussion is based on Figure 4-12 on page 90, Figure 4-13 on page 91, and the discussion in Section 4.4.6.2 on page 85.

1 Refer to Section 4.4.6.2 on page 85, Step 1.

2 Refer to Section 4.4.6.2 on page 85, Step 2.

For each LOB data column, the FD:OCA placeholder indicator flag is set on to indicate that the column data will be externalized and flow in an EXTDTA object. The column data will flow in this manner unless the application overrides the descriptor using a SQLDA with the FETCH request.

Since the answer set contains LOB data columns, no query data is returned at this time. The first row is returned with the first CNTQRY command, allowing the application to specify an overriding SQLDA with the FETCH request.

3 Refer to Section 4.4.6.2 on page 85, Step 3.

The parameter *rtnextdta* must be specified on each CNTQRY if a value other than the default is desired.

If the application wishes to receive a LOB column in a form other than as data, then it specifies a SQLDA descriptor on the FETCH request and the application requester must create an OUTOVR to send to the application server. The application server uses the overriding OUTOVR to determine the format of the data returned. If no OUTOVR is sent to the application server, the application server fetches the data using the server description of the data as given in the QRYDSC returned by the target with the OPNQRYRM.

The application requester sends the CNTQRY command and the optional OUTOVR object to the application server.

4 Refer to Section 4.4.6.2 on page 85, Step 4.

The application server receives and processes the CNTQRY. It processes the OUTOVR, if sent, to specify the format in which the relational database is to fetch the row or rows.

The application server creates a QRYDTA reply object as in Section 4.4.6.2 on page 85, Step 4.

In addition, LOB data columns are returned as follows: All LOB locators are returned in the QRYDTA object. Each LOB column is represented by an FD:OCA placeholder in the QRYDTA and the LOB value bytes are returned in an associated EXTDTA object following the QRYDTA. If more than one EXTDTA is returned, they are returned in the order that their corresponding FD:OCA placeholders occur in the QRYDTA object.

The application server sends the QRYDTA object to the application requester and returns the EXTDTA objects according to the *rtnextdta* option.

In this discussion, the *rtnextdta* value is *rtnextall*:

The QRYDTA object is returned along with the EXTDTA objects associated with all complete rows contained in the QRYDTA object. The EXTDTAs associated with a partial row are not returned until the complete row is returned in the next QRYDTA object.

The application requester may also request that extra query blocks be returned by means of the *maxblkext* value. Even if extra query blocks are requested by the application requester, the application server is not required to return any extra query blocks nor is it required to return the number requested. The application server may choose to return extra query blocks in some cases, but not in others. For example, the application server may choose not to return extra query blocks if there are LOBs in the answer set. If the application server does support the return of extra query blocks, however, it must adhere to certain rules. For example, extra query blocks are not to be sent if the extra blocks do not complete a row and do not contain a complete row.

If the application server returns extra query blocks when there are LOBs in the answer set, the following applies:

- If *maxblkext* is zero, then no extra query blocks are returned. The query is suspended after the last EXTDTA for the last complete row in the QRYDTA object is returned to the source system.
- If *maxblkext* is *n*, where *n* is a positive value, then the first extra query block to be returned is sent, followed by the EXTDTA objects associated with complete rows contained in the extra query block. If the application server determines that it will send extra query blocks for the query, then it sends the first extra query block followed by the EXTDTAs associated with complete rows contained in the extra query block. This repeats for all *n* extra query blocks to be sent or until the answer set is complete.
- If *maxblkext* is −1, the application server returns a query block of answer set data, followed by the EXTDTAs associated with the query block. If the application server determines that it will send extra query blocks, then it returns the entire answer set, including all QRYDTA objects and their associated EXTDTA objects.

5 Refer to Section 4.4.6.2 on page 85, Step 5.

The application requester receives the QRYDTA block. The first row data received in the QRYDTA are mapped to the application's host variables. Data represented in the row by FD:OCA placeholders are to be obtained from EXTDTAs according to the *rtnextdta* specified.

In this discussion, the *rtnextdta* value is *rtnextall*:

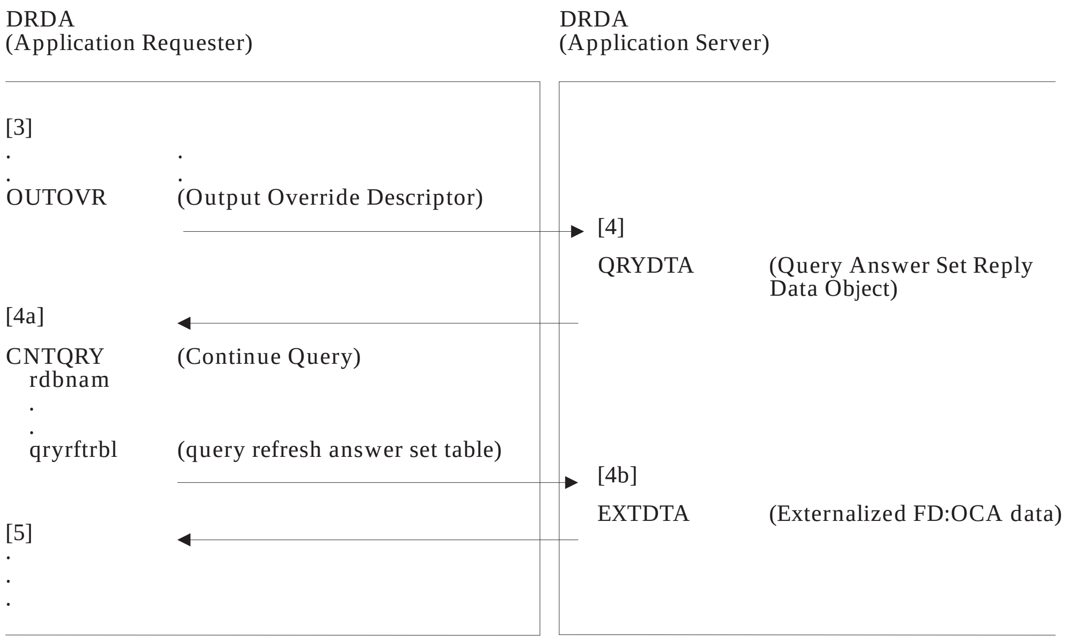
The application requester returns data to the application with each application FETCH request, obtaining non-FD:OCA placeholder data from the QRYDTA block and LOB value bytes from EXTDTAs associated with the FD:OCA placeholders in the row. If extra blocks were returned, processing continues with the extra query blocks. If no more complete rows are contained in the query blocks returned by the application server, the application requester formats a CNTQRY command and sends it to the application server.

n Refer to Section 4.4.6.2 on page 85, Step *n*. If LOB data is returned with the QRYDTA, then even though the end of the query data is reached, the ENDQRYRM is not returned until the next CNTQRY, in order to maintain the chaining rules for the QRYDTA and EXTDTA objects being returned.

n+1 Refer to Section 4.4.6.2 on page 85, Step *n+1*.

Limited Block Protocol (rtnextdta=rtnextrow)

This discussion is based on Figure 4-12 on page 90, Figure 4-13 on page 91, and the discussion in Section 4.4.6.3 on page 90. Additional commands required for this case are included in Figure 4-14.


DRDA
(Application Requester)
DRDA
(Application Server)
[3]
OUTOVR (Output Override Descriptor)
[4]
QRYDTA (Query Answer Set Reply Data Object)
[4a]
CNTQRY (Continue Query)
rdbnam
qryrftrbl (query refresh answer set table)
[4b]
EXTDTA (Externalized FD:OCA data)
[5]

[4a] and [4b] repeat until the LOBs are returned for the base data returned in [4]. The flow then proceeds to [5] to get the next set of base data. If any data is returned, [4a] and [4b] are repeated again.

Figure 4-14 Limited Block Protocol Query Processing (with LOB Data, rtnextrow)

1 Refer to **Limited Block Protocol (rtnextdta=rtnextall)** on page 92, Step 1.

2 Refer to **Limited Block Protocol (rtnextdta=rtnextall)** on page 92, Step 2.

3 Refer to **Limited Block Protocol (rtnextdta=rtnextall)** on page 92, Step 3.

4 Refer to **Limited Block Protocol (rtnextdta=rtnextall)** on page 92, Step 4.

In this discussion, the *rtnextdta* value is *rtnextrow*:

The initial CNTQRY command for the query returns one or more QRYDTA objects containing one or more base data rows. No EXTDTAs associated with the base data are returned. See Step 4a and Step 4b for the flow which returns the associated EXTDTA objects.

4a The application requester returns data to the application for the first application FETCH request after obtaining both the base data and the externalized data for the row. The application obtains base data from the QRYDTA object.

If the base data row is complete and all of its LOB data columns are trivial (nullable columns that are null or columns with zero placeholder values), then there is no pending LOB data for this row and the application rueqester returns to the application with the fetched base data.

If the base data row is complete and it has non-trivial LOB data columns (nullable columns that are not null or columns with non-zero placeholder values), the application requester sends a CNTQRY command to the application server to obtain the associated EXTDTA objects.

If the base data row is not complete, this step reverts to Step 3 in Section 4.4.6.2 on page 85, where the CNTQRY command is sent to obtain or complete the base data for the next row.

4b The application server receives the CNTQRY command.

For the next previously sent complete row of base data, the application server returns the associated EXTDTA objects for the LOB columns pending for that base row.

If there are no more pending LOB data columns for previously sent rows, this step reverts to Step 4 in Section 4.4.6.2 on page 85, where the CNTQRY command causes the application server to obtain or complete the base data for the next row.

Steps 4a and 4b are repeated until the application requester processes all the complete rows in the received QRYDTA objects or until the application does not fetch any more rows.

5 Refer to Section 4.4.6.2 on page 85, Step 5.

n Refer to Section 4.4.6.2 on page 85, Step *n*.

n+1 Refer to Section 4.4.6.2 on page 85, Step *n+1*.

4.4.7 Executing a Bound SQL Statement

This section describes the DDM commands and replies that flow during the execution of SQL statements that have been bound by the bind process or the PRPSQLSTT command. Section 4.4.7.1 on page 97 describes the commands and replies that flow in most instances. Section 4.4.7.2 on page 100 describes the commands and replies that flow for an SQL statement that invokes a stored procedure which returns result sets.

If connection is between an application server and database server, any new or changed special register settings must be sent using the EXCSQLSET command prior to activating or processing queries. The EXCSQLSET command is recommended to be chained next SQL command.

4.4.7.1 Executing Ordinary Bound SQL Statements

Figure 4-15 indicates the DDM commands and replies that flow during the execution of the majority of SQL statements that can be bound by the bind process or the PRPSQLSTT command. The usual result is that the application server makes the expected changes in the relational database (within the unit of work) after the indicated bound SQL statement has successfully executed. For a description of the commands and replies that flow for an SQL statement that invokes a stored procedure which returns result sets, refer to Section 4.4.7.2 on page 100.

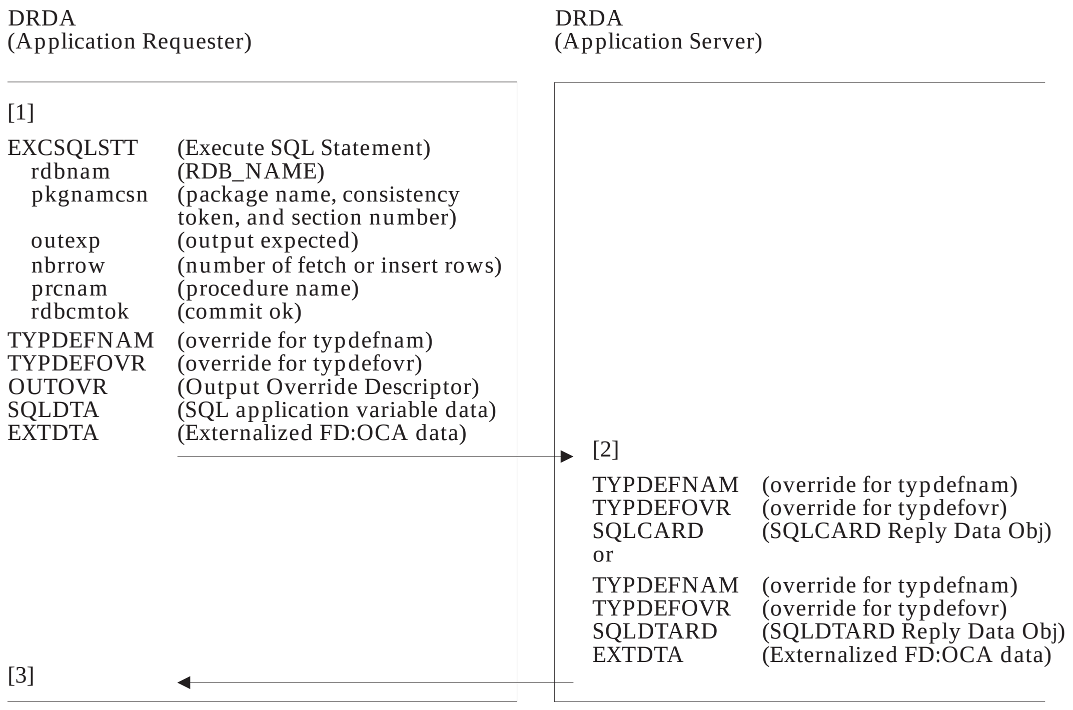

DRDA
(Application Requester)
DRDA
(Application Server)
[1]
EXCSQLSTT (Execute SQL Statement)
rdbnam (RDB_NAME)
pkgnamcsn (package name, consistency token, and section number)
outexp (output expected)
nbrrow (number of fetch or insert rows)
prcnam (procedure name)
rdbcmtok (commit ok)
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
OUTOVR (Output Override Descriptor)
SQLDTA (SQL application variable data)
EXTDTA (Externalized FD:OCA data)
[2]
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLCARD (SQLCARD Reply Data Obj)
or
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLDTARD (SQLDTARD Reply Data Obj)
EXTDTA (Externalized FD:OCA data)
[3]

Figure 4-15 DRDA Flows: Executing a Bound SQL Statement

The following is a discussion of the operations and functions the application requester and the application server perform. This volume provides a brief description of some of the parameters for the DDM commands. See the DDM Reference for a detailed description of the parameters.

1 After the application requester and the application server have established proper connection (described in Figure 4-2 on page 55), prebound SQL statements referenced in a package in a remote relational database can be executed. (See Section 4.4.3 on page 67 for a discussion of the DRDA flows needed to perform the bind.) Other DRDA flows can precede or follow the execution of the prebound SQL statement referenced in the package and be part of the same unit of work.

The application requester that is acting as the agent for the application performing the execute SQL statement function creates the Execute SQL Statement (EXCSQLSTT) command by providing the correct package name, consistency token, and section number in the *pkgnamcsn* parameter. It also indicates in the *outexp* parameter whether or not it expects output to be returned within an SQLDTARD reply data object as a result of the execution of the SQL statement. The optional *nbrrow* parameter[31] indicates the number of rows to insert if the operation of the EXCSQLSTT is an SQL multi-row insert. The optional *rdbcmtok* parameter informs the RDB whether or not to process commit and rollback operations. The optional *prcnam* parameter identifies the stored procedure to be executed at the application server. The application requester also puts any application variable values and their descriptions in the SQLDTA command data object. All data types for host variables associated with a CALL or other statement that invokes a stored procedure must be nullable when they flow on the wire, so if a data type is non-nullable, it must be turned into the nullable form of the data type by the application requester prior to sending to the application server.

All host variables associated with the parameter list of a stored procedure must be reflected with a null indication or data in the SQLDTA.

If a CALL or other statement that invokes a stored procedure specifies the procedure name using a host variable, then the *prcnam* parameter of the EXCSQLSTT specifies the procedure name value. The procedure name value is not duplicated in any SQLDTA command data object that might also flow with the EXCSQLSTT.

If the CALL or other statement that invokes a stored procedure does not specify the procedure name using a host variable, then the value specified by the *prcnam* parameter, if present, must match the procedure name value contained within the section identified by *pkgnamcsn*. It sends the command and command data to the application server.

The application requester may send LOB data as input host variables in the SQLDTA that accompanies an EXCSQLSTT or as input parameters in the SQLDTA for an EXCSQLSTT statement that is a CALL to a stored procedure. For each input host variable that is a LOB data type, an FD:OCA placeholder is placed in SQLDTA and the corresponding value bytes are flowed in an EXTDTA following the SQLDTA. The EXTDTAs flow in the order that the FD:OCA placeholders occur in the associated SQLDTA.

Note: For a stored procedure, output parameters that are LOBs are sent to the application server along with input parameters. If an application wishes to avoid sending this data to the application server, it must explicitly clear the host variable before making the call.

For an SQL statement that is not a stored procedure call, output may or may not be expected with the execution of the statement. If the expected output includes LOB data, then the application requester must send an OUTOVR object to the application server if the application wishes to receive a LOB locator in place of the actual data.

For an SQL statement that is a stored procedure call, if any of the output parameters is a LOB type, then the SQLDTA describes the desired format of the output. An OUTOVR object is not sent in this case and is rejected by the target system if it is sent.

2 The application server receives and processes the EXCSQLSTT command and creates an SQLCARD reply data object or SQLDTARD reply data object. The requested statement executes with the input variable values passed with the command, the results are reflected in the referenced database manager (within the scope of the unit of work), and an SQLCARD reply data object is returned. If errors occur during the execution of the statement, the referenced database manager remains unchanged, and the SQLCARD reply data object contains an indication of the error condition.

31. This parameter is not supported in DRDA Level 1.

If the execution of the SQL statement (a single row SELECT, statement that invokes a stored procedure, or SET statement) generates output data, the application server returns this data in the SQLDTARD reply data object.

All host variables associated with the parameter list of a stored procedure must be reflected with a null indication or data in the SQLDTARD. The application server also returns the SQLCA in the SQLDTARD, ahead of the data, indicating the normal completion of SQL statement execution.

Note: If the execution of the statement generates output data, which was not expected (indicated on the *outexp* parameter), then the application server sends the SQLCARD to the application requester indicating an error and does not send any output data.

If the section identified by *pkgnamcsn* exists in the package identified by *pkgnamcsn*, but the section is not associated with a stored procedure, then the use of *prcnam* with *pkgnamcsn* is invalid and the application server returns CMDCHKRM to the application requester.

If the executed SQL statement is either a COMMIT or ROLLBACK, see Section 4.4.12.1 on page 117 and Section 4.4.12.2 on page 120 for a description of commit and rollback processing in DRDA.

If any data is to be returned by the application server, the application server creates an SQLDTARD. For each output host variable that is a LOB data type, an FD:OCA placeholder is placed in the SQLDTARD and the corresponding value bytes are flowed in a EXTDTA following the SQLDTARD. The EXTDTAs flow in the order that the FD:OCA placeholders occur in the associated SQLDTARD.

If an OUTOVR object is received, it will be used to format the output, if output is expected and the SQL statement is not a stored procedure call. For a stored procedure call, the OUTOVR object is rejected. For an SQL statement that does not return output, the OUTOVR object is rejected.

3 For a normal completion, the application requester returns to the application with the successful indication. The application requester also returns any data in the SQLDTARD reply data object to the application.

At this point, the application/application requester can continue with additional defined DRDA flows.

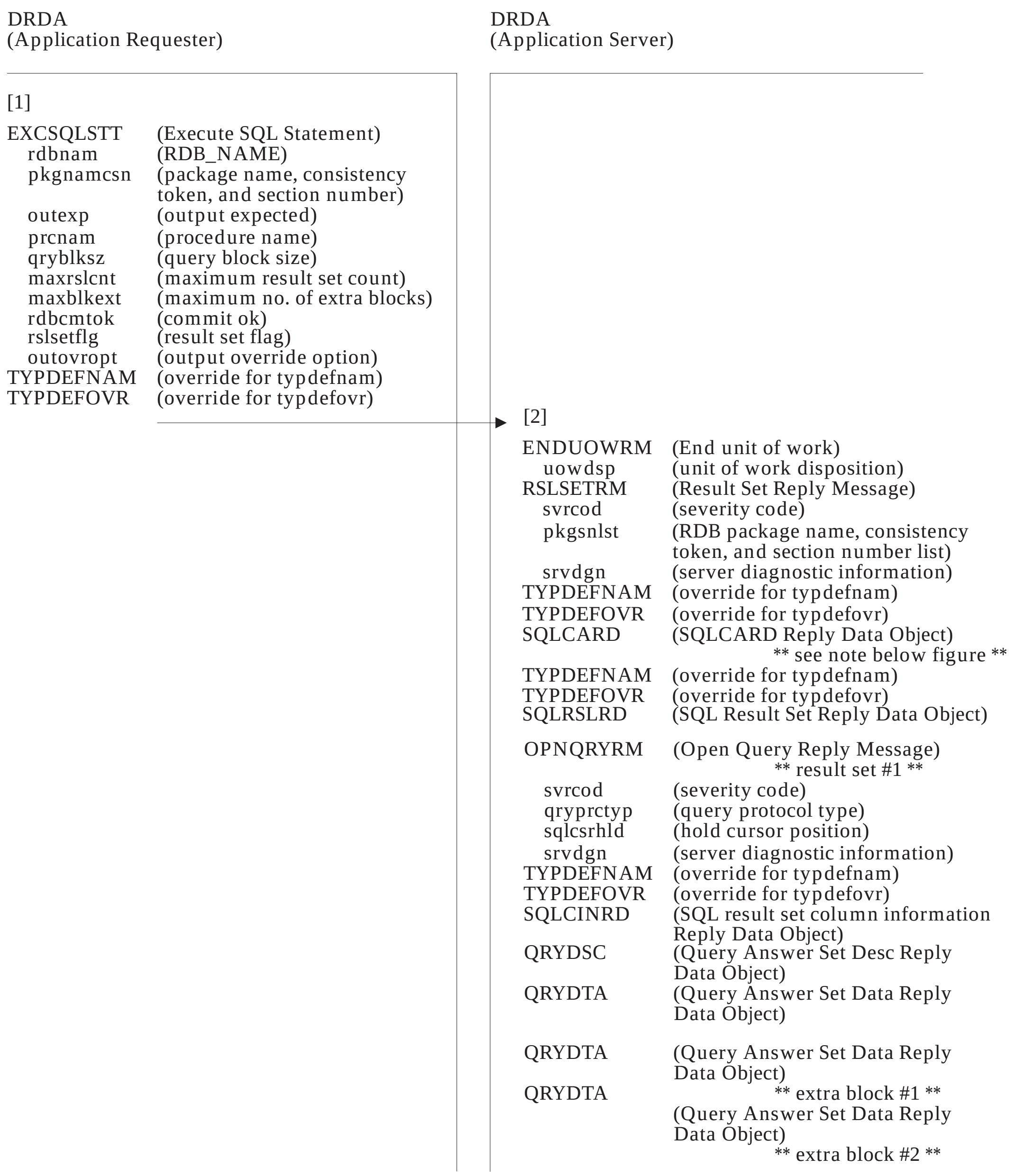
If the SQLCARD reply data object that the application server returned to the application requester indicates that the EXCSQLSTT command was not successful, the application requester returns an exception to the application that is attempting to execute the SQL statement.

4.4.7.2 Invoking a Stored Procedure that Returns Result Sets

Figure 4-16 on page 101 indicates the DDM commands and replies that flow during the execution of an SQL statement, previously bound by the bind process or the PRPSQLSTT command, that invokes a stored procedure which returns result sets. These flows produce the desired effect needed to satisfy an application program that executes a stored procedure and FETCHes the rows from result sets generated by the execution of that stored procedure. The application server ships the answer set data to the application requester and the application requester then returns the row data of the answer sets to the application in whatever order the application requests them. This example assumes that the application requester desires the names for columns within results sets and is capable of processing answer set data returned in the response to EXCSQLSTT. Although this example illustrates a stored procedure that returns two result sets, DRDA (using SQLAM Level 5) supports the return of any number of result sets. The example also assumes that the stored procedure has been defined with the *commit on return* attribute and the result set cursors within the stored procedure are defined with the HOLD option. Although result set cursors can return data according to the rules for either the fixed row protocol or the limited block protocol, the example only shows the use of the limited block protocol rules. Since result set cursors are unambiguously read-only, generally the rules for limited block protocol can be used, so this example shows the predominate scenario. This choice can be superseded, for example, if the EXCSQLSTT for the call statement specifies an *outovropt* of *outovrany*, causing any result sets that return LOB output values to be returned using fixed row protocol rules.

The application server sends the row data of the answer sets grouped into blocks following the rules for limited block protocol and according to the options specified by the application requester on the EXCSQLSTT and CNTQRY commands. For details on the description of answer set blocks and how they are supported using the limited block protocol, see the terms QRYBLK, QRYBLKCTL, QRYBLKSZ, LMTBLKPRC, MAXBLKEXT, and MAXRSLCNT in the DDM Reference. Also see the rules for query processing in Section 7.19 on page 310.

The following example describes the various flows that show how the application server returns row data of the answer sets to the application requester and how the application requester requests more row data of the answer sets from the application server, if needed.



DRDA
(Application Requester)

[1]

EXCSQLSTT	(Execute SQL Statement)
rdbnam	(RDB_NAME)
pkgnamcsn	(package name, consistency token, and section number)
outexp	(output expected)
prcnam	(procedure name)
qryblksz	(query block size)
maxrslcnt	(maximum result set count)
maxblkext	(maximum no. of extra blocks)
rdbcmtok	(commit ok)
rslsetflg	(result set flag)
outovropt	(output override option)
TYPDEFNAM	(override for typdefnam)
TYPDEFOVR	(override for typdefovr)

DRDA
(Application Server)

[2]

ENDUOWRM	(End unit of work)
uowdsp	(unit of work disposition)
RSLSETRM	(Result Set Reply Message)
svrcod	(severity code)
pkgsnlst	(RDB package name, consistency token, and section number list)
srvdgn	(server diagnostic information)
TYPDEFNAM	(override for typdefnam)
TYPDEFOVR	(override for typdefovr)
SQLCARD	(SQLCARD Reply Data Object) ** see note below figure **
TYPDEFNAM	(override for typdefnam)
TYPDEFOVR	(override for typdefovr)
SQLRSLRD	(SQL Result Set Reply Data Object)
OPNQRYRM	(Open Query Reply Message) ** result set #1 **
svrcod	(severity code)
qryprctyp	(query protocol type)
sqlcsrhld	(hold cursor position)
srvdgn	(server diagnostic information)
TYPDEFNAM	(override for typdefnam)
TYPDEFOVR	(override for typdefovr)
SQLCINRD	(SQL result set column information Reply Data Object)
QRYDSC	(Query Answer Set Desc Reply Data Object)
QRYDTA	(Query Answer Set Data Reply Data Object)
QRYDTA	(Query Answer Set Data Reply Data Object) ** extra block #1 **
QRYDTA	(Query Answer Set Data Reply Data Object) ** extra block #2 **

Figure 4-16 DRDA Flows: Executing a Stored Procedure (Part 1)

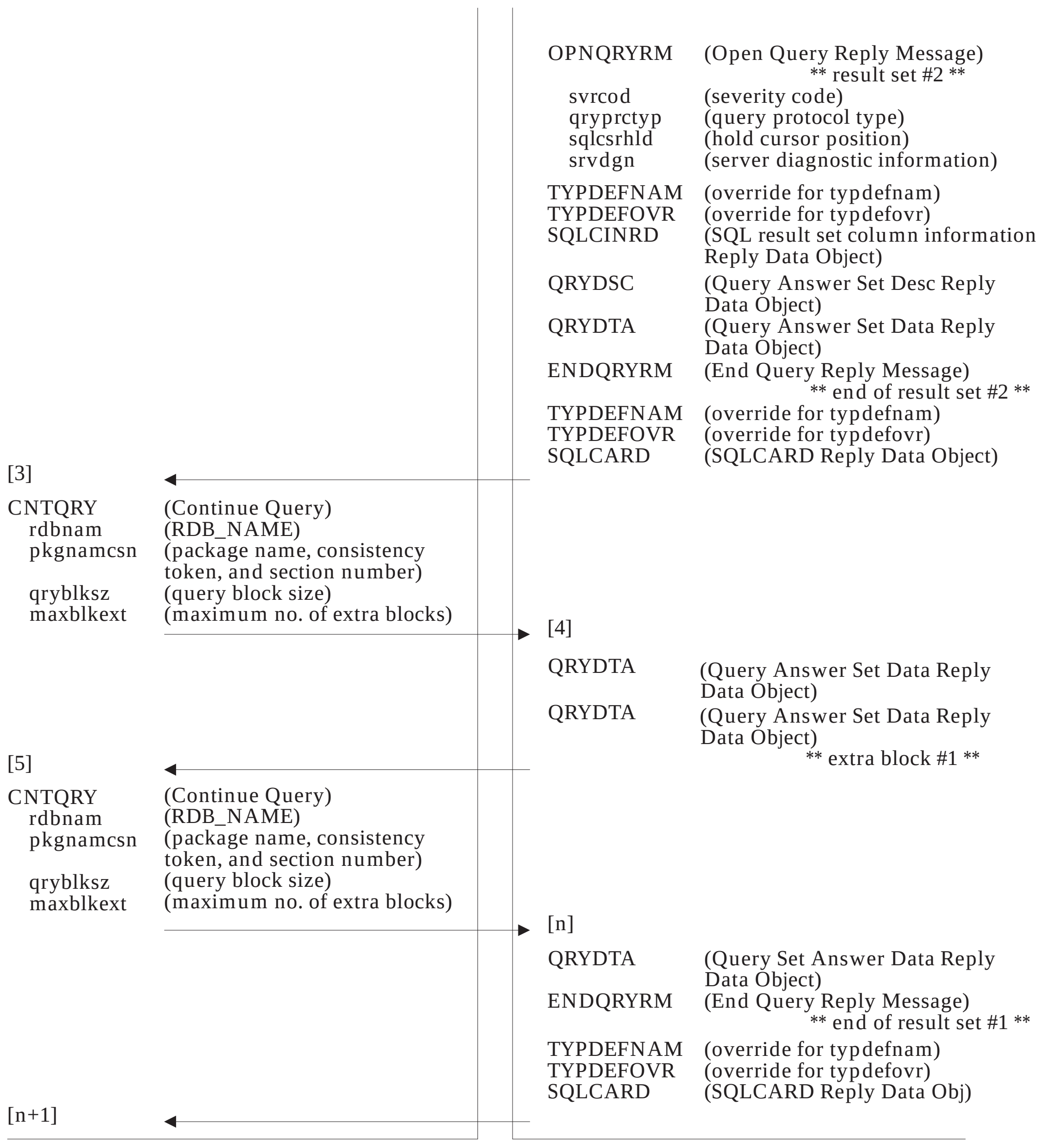

OPNQRYRM (Open Query Reply Message)
** result set #2 **
svrcod (severity code)
qryprctyp (query protocol type)
sqlcsrhld (hold cursor position)
srvdgn (server diagnostic information)
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLCINRD (SQL result set column information Reply Data Object)
QRYDSC (Query Answer Set Desc Reply Data Object)
QRYDTA (Query Answer Set Data Reply Data Object)
ENDQRYRM (End Query Reply Message)
** end of result set #2 **
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLCARD (SQLCARD Reply Data Object)
[3]
CNTQRY (Continue Query)
rdbnam (RDB_NAME)
pkgnamcsn (package name, consistency token, and section number)
qryblksz (query block size)
maxblkext (maximum no. of extra blocks)
[4]
QRYDTA (Query Answer Set Data Reply Data Object)
QRYDTA (Query Answer Set Data Reply Data Object)
** extra block #1 **
[5]
CNTQRY (Continue Query)
rdbnam (RDB_NAME)
pkgnamcsn (package name, consistency token, and section number)
qryblksz (query block size)
maxblkext (maximum no. of extra blocks)
[n]
QRYDTA (Query Set Answer Data Reply Data Object)
ENDQRYRM (End Query Reply Message)
** end of result set #1 **
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLCARD (SQLCARD Reply Data Obj)
[n+1]

Figure 4-17 DRDA Flows: Executing a Stored Procedure (Part 2)

Note: If there are host variables in the parameter list of the SQL statement that invoked the stored procedure, then an SQLDTA command data object flows from the application requester to the application server on the EXCSQLSTT command and an SQLDTARD reply data object, rather than an SQLCARD, flows from the application server to the application requester within the summary component of the response to the EXCSQLSTT command.

The following is a discussion of the operations and functions the application requester and the application server perform. This volume provides a brief description of some of the parameters for the DDM commands. See the DDM Reference for a detailed description of the parameters.

Although Figure 4-16 on page 101 and Figure 4-17 assume that there are no LOBs in any of the result sets, the following discussion indicates where LOB-related processing occurs. Refer to

Section 4.4.6 on page 76 for additional discussions of LOB-related processing for query result sets.

1 After the application requester and the application server have established proper connection (described in Figure 4-2 on page 55), prebound SQL statements referenced in a package in a remote relational database can be executed. (See Section 4.4.3 on page 67 for a discussion of the DRDA flows needed to perform the bind.) Other DRDA flows can precede or follow the execution of the prebound SQL statement referenced in the package and be part of the same unit of work.

The application requester that is acting as the agent for the application performing the execute SQL statement function creates the Execute SQL Statement (EXCSQLSTT) command by providing the correct package name, consistency token, and section number in the *pkgnamcsn* parameter. The application requester sets the *outexp* parameter to TRUE or FALSE depending on whether it expects an SQLDTARD reply data object to be returned within the response to the EXCSQLSTT. The optional[32] *prcnam* parameter identifies the stored procedure to be executed at the application server. The application requester specifies the query block size for the reply data objects and reply messages that the application server can return for this command in the *qryblksz* parameter.

Note: The block size specified in *qryblksz* must be equal to or greater than 512 bytes and equal to or less than 32,767 bytes. If not, the application server returns the VALNSPRM reply message and the application server does not execute the command.

The application requester specifies the maximum number of result sets the application requester is capable of receiving in the *maxrslcnt* parameter. For this example, assume that the value of the *maxrslcnt* parameter is two. The application requester specifies the maximum number of extra data blocks that the application requester is capable of receiving per result set in the *maxblkext* parameter. For this example, assume that the value of the *maxblkext* parameter on EXCSQLSTT is two. The application requester is also responsible for putting any application variable values and their descriptions in the SQLDTA command data object. For this example, assume that there are no application variable values. Thus, in this instance, the application requester does not include an SQLDTA object as command data on the EXCSQLSTT command. The application requester specifies whether it desires the application server to return name, label, and comment information for the columns of result sets and whether it desires the application server to return result answer set data in the response to EXCSQLSTT in the *rslsetflg* parameter. For this example, assume that the application requester desires the return of result set column names and answer set data. The *rdbcmtok* parameter is set to TRUE in this example to allow the server to process the commit operation that occurs as a result of the stored procedure call. The application requester sends the command and command data to the application server.

2 The application server receives and processes the EXCSQLSTT command. The *maxrslcnt* parameter limits the number of result sets that the application server may return to two and indicates that the application requester expects result set data to be returned by this command. Thus, the application server assumes the use of limited block protocols. The *maxblkext* parameter limits the number of extra data blocks that

32. SQLAM Level 5 is required to support this parameter.

the application server may return per result set to two. The *qryblksz* parameter determines the size of each query block. The *rslsetflg* parameter indicates that the application requester is capable of processing answer set data in the response to EXCSQLSTT.

The application server invokes the stored procedure. The stored procedure executes and generates result sets in the order required by the logic and state information of the stored procedure. In this sample flow, the stored procedure generates two result sets. Before the execution of the stored procedure completes, the stored procedure specifies the order in which the application server is to return the result sets to the application requester.

The execution of the stored procedure completes. Since the stored procedure was defined with the *commit on return* attribute and *rdbcmtok* was specified as TRUE in the EXCSQLSTT command, the application server initiates commit processing. When commit processing completes successfully ENDUOWRM with *uowdsp* set to committed becomes the first part of the response. The response continues with a summary component and at most *m* result set components, where *m* is the value of the *maxrslcnt* parameter, specified by the application requester on the EXCSQLSTT command. In this sample flow, the value of the *maxrslcnt* parameter is two and the number of result sets is also two. The result set components follow the summary component in the response and are arranged in the order specified by the stored procedure for the return of result sets to the application requester.

The application server constructs the summary component, which consists of an RSLSETRM reply message, an SQLCARD reply data object, and an SQLRSLRD reply data object. The RSLSETRM reply message contains a *pkgsnlst* parameter that lists the *pkgnamcsn* values for the result sets in the order in which the application server will return the result sets to the application requester.[33] The SQLCARD reply data object conveys information about the success of the SQL statement that invoked the stored procedure. The SQLRSLRD reply data object sequences the locator value, name information, and the number of rows for the result sets in the order in which the application server will return the result sets to the application requester.

The application server then constructs the result set components for the result sets generated by the execution of the stored procedure. Each result set component contains at least the OPNQRYRM, the SQLCINRD, and the FD:OCA description of the data (QRYDSC). The block containing the end of the FD:OCA description may be completed, if room exists, with answer set data. Additional blocks of answer set data may also be chained to the block containing the end of the FD:OCA description, up to the maximum number of extra blocks of answer set data specified by the application requester in the *maxblkext* parameter of the EXCSQLSTT command.

If any cursors in the stored procedure will result in LOB data being returned in the answer set, then the application server does not return any QRYDTA for the cursor until the application at the application requester issues a FETCH request. So, for each

33. At the time the application server constructs the OPNQRYRM reply message for a result set, the application server also associates a *pkgnamcsn*, locator value, and name with the result set. Each *pkgnamcsn* value identifies a section in a package at the application server that is assigned to the result set. The locator value is a unique identifier for the result set that allows the application to describe, fetch rows from, or declare a cursor on the associated result set. The name conveys the semantic of the result set and is returned to the application so that the application can associate the result set with application logic for processing the result set.

such cursor returned by the stored procedure, the application server returns:

- OPNQRYRM
- SQLCINRD
- QRYDSC

In this sample flow, the response to the EXCSQLSTT command consists of five blocks. The first block contains the summary component. The second block contains the OPNQRYRM reply message, the SQLCINRD reply data object, the QRYDSC reply data object, and a QRYDTA reply data object for the first result set. The third and fourth blocks each contain an additional QRYDTA reply data object for the first result set. The fifth block contains an OPNQRYRM reply message, the SQLCINRD reply data object, the QRYDSC reply data object, a QRYDTA reply object, an ENDQRYRM reply message, and an SQLCARD for the second result set. For each OPNQRYRM reply message, the value of the *qryprctyp* parameter is LMTBLKPRC. The application server sends the response to the application requester.

3 The application requester receives the ENDUOWRM reply message, RSLSETRM reply message, the SQLCARD reply data object, and the SQLRSLRD reply data object from the application server. The receipt of ENDUOWRM informs the application requester that a commit operation occurred at the application server and the current unit of work has terminated. As a result of this the application requester may have to perform additional processing. See Section 4.4.12.2 on page 120 for details. The receipt of the RSLSETRM informs the application requester that information about result sets follows the SQLCARD. The application requester returns the execution results for the SQL statement that invoked the stored procedure (the information content of the SQLCARD) to the application at the application requester. If the number of result sets returned by the application server exceeds the limit (that is, the MAXRSLCNT parameter value) that the application requester is capable of receiving, the number of extra blocks of answer set data returned by the application server for a result set exceeds the limit (that is, the MAXBLKEXT parameter value) that the application requester is capable of receiving, the number of result set entries within the SQL Result Set Reply Data object (SQLRSLRD) returned by the application server does not match the number of Open Query Complete reply messages (OPNQRYRMs) returned by the application server, or the number of result set entries within the SQL Result Set Reply Data object (SQLRSLRD) returned by the application server does not match the number of entries within the RDB Package Name, Consistency Token, and Section Number List (PKGSNLST) returned by the application server, then the application requester may return SQLSTATE X‘58008’ or SQLSTATE X‘58009’ to the application.

The application requester receives the query blocks for each result set from the application server. The application requester associates each query block with the *pkgnamcsn*, locator value, and name of its result set and then stores the description and answer set data associated with each result set for subsequent FETCH by the application at the application requester.

No further flows are required between the application requester and the application server for the transmission of additional answer set data unless the application issues a FETCH that cannot be satisfied by the QRYDTA reply data object already stored at the application requester for a result set. This sample flow assumes that the client application at the application requester does issue a FETCH for the first result set that the application requester cannot satisfy.

If the application at the AR issues a FETCH using a descriptor, then the application requester may optionally format an OUTOVR object and flow it to the application server with the CNTQRY. The OUTOVR is required only if the application wishes to receive LOB columns in a format other than as LOB data.

When the application performs a FETCH for the first result set and a complete row is no longer available in the QRYDTA reply data object, the application requester creates a CNTQRY command that specifies the *pkgnamcsn* value returned for that result set in the *pkgsnlst* parameter of the RSLSETRM reply message. The application requester may also specify different values for the *qryblksz* and *maxblkext* parameters of the CNTQRY command from those specified on the EXCSQLSTT command. For this sample flow, assume that the value of the *maxblkext* parameter on the CNTQRY command is one.

4 The application server receives the CNTQRY command and identifies the result set associated with the CNTQRY request through the section number contained within the *pkgnamcsn* parameter. If it has a partial row from the EXCSQLSTT command, it places the partial row in the QRYDTA reply data object. If it does not have a partial row, then it retrieves the next data row from the answer set and places it in the QRYDTA reply data object along with an SQLCA. The partial row or next row may span query blocks. The block containing the end of the partial row or next row may be completed, if room exists, with additional answer set data. Additional blocks of answer set data may also be chained to this block of answer set data up to the maximum number of extra blocks of answer set data specified by the application requester in the *maxblkext* parameter of the CNTQRY command.

If the application server receives a CNTQRY with an OUTOVR object, then it either accepts or rejects the OUTOVR object depending on the *outovropt* value on the OPNQRY or EXCSQLSTT command. If it accepts the OUTOVR, it returns the output data in the format given by the override descriptors.

Columns that will be returned as LOB data flow in the QRYDTA as FD:OCA placeholders. The data values themselves flow in EXTDTA objects after the QRYDTA object containing the associated row.

In this sample flow, the response to the CNTQRY command consists of two blocks. Both blocks contain QRYDTA reply data objects containing answer set data from the first result set. The application server sends the query blocks to the application requester.

5 When the application requester receives the QRYDTA reply data object, it passes the row data that spanned the previous query block and the one just received (if any) to the application. The application requester maps the row data it received in the QRYDTA reply data object to the application's host variables (in response to the previously unsatisfied FETCH operation) based on the description passed in the QRYDSC reply data object within the response to the EXCSQLSTT command.

When the application requests the next row from the application requester, the application requester maps the next row from the QRYDTA reply data object to the application's host variables.

When a complete row of the first result set is no longer available in the query block, the application requester creates a CNTQRY command that specifies the *pkgnamcsn* value returned for that result set in the *pkgsnlst* parameter of the RSLSETRM reply message. The application requester may also specify different values on the *qryblksz* and *maxblkext* parameters for the CNTQRY command than those specified on the

EXCSQLSTT command.

Steps 4 and 5 are repeated until the application server returns the QRYDTA reply data object with all of the last row of the answer set for the first result set to the application requester, or until the application does not request any more rows.

n The application server receives the CNTQRY command and identifies the result set associated with the CNTQRY request through the section number contained within the *pkgnamcsn* parameter. If it has a partial row from the last CNTQRY command, it places the partial row in the QRYDTA reply data object. If it does not have a partial row, then it retrieves the next data row from the answer set and places it in the QRYDTA reply data object along with an SQLCA. The partial row or next row may span query blocks. The application server continues to retrieve additional rows of the answer set and to place them in the QRYDTA reply data object until it has retrieved the last row. After it has placed the last row of the answer set (which completes the last QRYDTA reply data object) in the query block, the application server generates an ENDQRYRM reply message and an SQLCARD reply data object and places them in the query block.

The application server then closes the query and sends the last query block to the application requester.

n+1 When the application requester receives the reply block, it passes the row data that spanned the previous QRYDTA reply data object and the one just received (if any) to the application. The application requester maps the row data it received in the QRYDTA reply data object to the application's host variables (in response to the previously unsatisfied FETCH operation) based on the description passed in the QRYDSC reply data object within the response to the EXCSQLSTT command.

When the application requests the next row from the application requester, the application requester maps the next row from the QRYDTA reply data object to the application's host variables.

When the application requester receives the ENDQRYRM reply message, it knows that the application server has processed the last row of answer data. It also knows that the application server has closed the query, so the application requester will not send any additional CNTQRY commands to the application server.

4.4.8 Preparing an SQL Statement

Figure 4-18 indicates the DRDA commands and replies that flow during the preparation of a single SQL statement. The usual result of this command is a prepared SQL statement in the indicated package that an EXCSQLSTT command can later (within the same unit of work) execute.

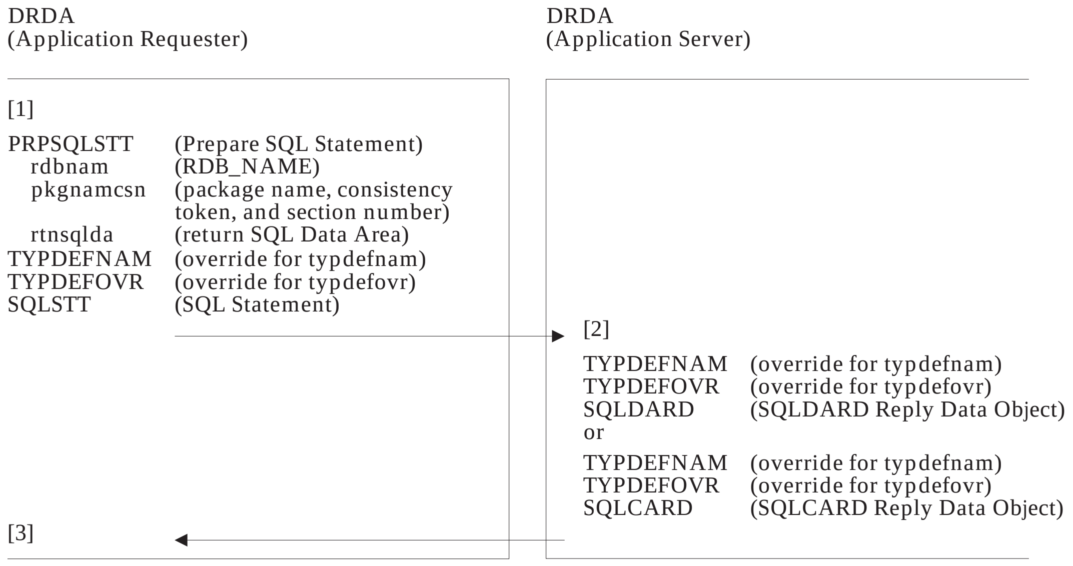

DRDA
(Application Requester)
DRDA
(Application Server)
[1]
PRPSQLSTT (Prepare SQL Statement)
rdbnam (RDB_NAME)
pkgnamcsn (package name, consistency token, and section number)
rtnsqlda (return SQL Data Area)
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLSTT (SQL Statement)
[2]
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLDARD (SQLDARD Reply Data Object)
or
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLCARD (SQLCARD Reply Data Object)
[3]

Figure 4-18 DRDA Flows: Preparing an SQL Statement

The following is a discussion of the operations and functions the application requester and the application server perform. This is a brief description of some of the parameters for the DDM commands. The DDM Reference provides a more detailed description of these parameters.

1 After the application requester and application server have established the proper connection (described in Figure 4-2 on page 55), the application server can prepare additional dynamic SQL statements (similar to bind), associated with a specified package, and later, within the same unit of work, the statement can execute.

Note: Other commands can precede or follow the preparation and execution of the SQL statement and be part of the same unit of work. The SQL statement can be executed as many times as needed within the same unit of work that it was prepared.

When the unit of work or the network connection terminates (normally or abnormally), the package no longer references the prepared statement, so the statement is no longer available for execution. However, when the unit of work is terminated with a COMMIT, the package still references the prepared statement for queries with the HOLD option in the DECLARE CURSOR statement, and the statement is still available for execution.

The application requester creates the Prepare SQL Statement (PRPSQLSTT) command by providing the correct package name, consistency token, and section number in the *pkgnamcsn* parameter. If the application requester needs a description of the row data that can be returned (when the statement being prepared is executed) as a result of a SELECT statement being prepared, then it indicates this in the *rtnsqlda* parameter. The application requester places the SQL statement to be prepared into the SQLSTT command data object and sends the command to the application server.

2 The application server receives and processes the PRPSQLSTT command and SQLSTT command data object and creates an SQLDARD reply data object or an SQLCARD reply data object. The application server prepares the requested SQL statement for later execution within this same unit of work. The application server performs a DESCRIBE (SQL verb) on the prepared statement, if indicated in the *rtnsqlda* parameter, and uses the returned row data descriptions to create an SQLDARD reply data object, which it returns to the application requester.

If the statement that the application server was preparing was not an SQL SELECT statement, then the SQLDARD reply data object will contain no SQLDA and a normal SQLCA. (The SQLDARD reply data object also contains the SQLCA, so the application server does not return the SQLCARD reply data object.)

If any errors occurred during the preparation of the SQL statement, the referenced package will not successfully prepare the new SQL statement, and the application server will return an SQLCA in either an SQLCARD reply data object or an SQLDARD reply data object (which will contain no SQLDA) indicating the error condition.

3 If an SQLCA that was found in the SQLCARD reply data object or the SQLDARD reply data object that the application server returned to the application requester indicates the PRPSQLSTT command was not successful, the application requester returns an exception to the application that is attempting to prepare the SQL statement.

Assuming it receives an SQLCARD reply data object or an SQLDARD reply data object indicating a normal completion of the PRPSQLSTT command, the application requester proceeds to return the successful indication to the application.

At this point, the application/application requester can continue with additional defined DRDA flows with the resulting database management changes being in the same unit of work, or it can execute the SQL statement that the process has prepared. A user can prepare multiple SQL statements and execute them within the same unit of work.

If the application requester is going to execute a prepared SQL statement next, it creates and sends an EXCSQLSTT command as described in step 1 of Figure 4-15 on page 97 or creates and sends an OPNQRY command as described in step 1 of Figure 4-9 on page 78.

4.4.9 Retrieving the Data Variable Definitions of an SQL Statement

Figure 4-19 indicates the DRDA commands and replies that flow during the retrieval of the data variable definitions associated with a bound SQL statement. The usual result of this command is the return of the definitions of the data variables that the desired SQL statement has referenced. The SQL statement can later be executed through an EXCSQLSTT command.

If connection is between an application server and database server, any new or changed special register settings must be sent using the EXCSQLSET command prior to activating or processing queries. The EXCSQLSET command is recommended to be chained next SQL command.

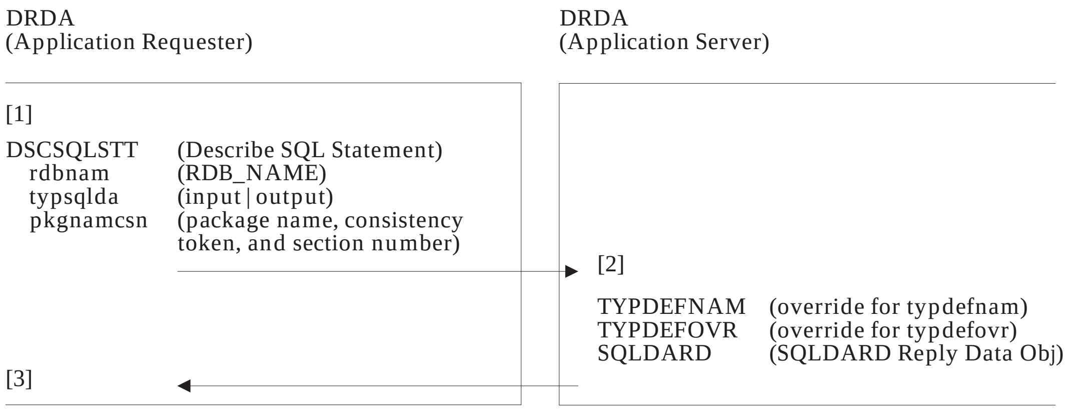

DRDA
(Application Requester)
DRDA
(Application Server)
[1]
DSCSQLSTT (Describe SQL Statement)
rdbnam (RDB_NAME)
typsqlda (input | output)
pkgnamcsn (package name, consistency token, and section number)
[2]
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLDARD (SQLDARD Reply Data Obj)
[3]

Figure 4-19 DRDA Flows: Describing a Bound SQL Statement

The following is a discussion of the operations and functions the application requester and the application server perform. This is a brief description of some of the parameters for the DDM commands. The DDM Reference provides a detailed description of these parameters.

1 After the application requester and the application server have established the proper connection (described in Figure 4-2 on page 55), it is possible to request the application server to provide either the description of the data variables that a particular SQL statement in a specific bound package references or to obtain definitions of the input parameters of a prepared statement.

The application requester creates the Describe SQL Statement (DSCSQLSTT) command by providing the correct package name, consistency token, and section number in the *pkgnamcsn* parameter. It then sends the command to the application server.

2 The application server receives and processes the DSCSQLSTT command. Then the application server creates an SQLDARD containing the requested data variable definitions for the indicated SQL statement and returns it to the application requester. (The SQLDARD reply data object also contains the SQLCA, so the application server does not return the SQLCARD reply data object.) If the application server found any errors while it described the SQL statement, the SQLDARD reply data object will contain an SQLCA, describing the error condition, and will not contain the data variable definitions. In either case, the application server returns the SQLDARD reply data object to the application requester.

Note: If the current unit of work had been abnormally terminated, then the application server would have returned an SQLCARD reply data object and an ABNUOWRM reply message instead of the SQLDARD reply data object.

3 If the application server returned an SQLDARD or SQLCARD reply data object to the application requester indicating the DSCSQLSTT command was not successful, the application requester returns an exception to the application that is attempting to describe the SQL statement.

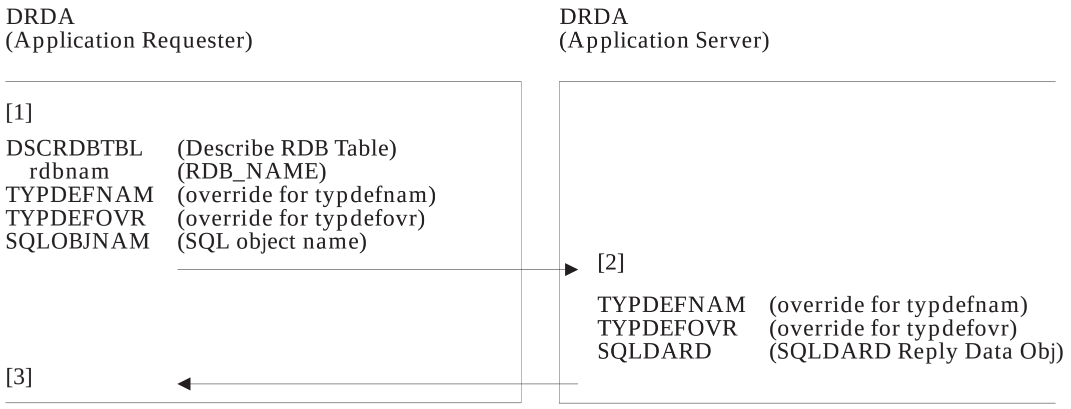
Assuming an SQLDARD reply data object, indicating a normal completion of the DSCSQLSTT command is received, the application requester proceeds to return the data variable definitions and the successful completion indication to the application.

At this point, the application/application requester can continue with additional defined DRDA flows with the resulting database management changes being in the same unit of work.

4.4.10 Executing a Describe Table SQL Statement

Figure 4-20 indicates the DRDA commands and replies that flow when executing a Describe Table SQL statement.

If connection is between an application server and database server, any new or changed special register settings must be sent using the EXCSQLSET command prior to activating or processing queries. The EXCSQLSET command is recommended to be chained next SQL command.



DRDA
(Application Requester)
DRDA
(Application Server)
[1]
DSCRDBTBL (Describe RDB Table)
rdbnam (RDB_NAME)
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLOBJNAM (SQL object name)
[2]
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLDARD (SQLDARD Reply Data Obj)
[3]

Figure 4-20 DRDA Flows: Describing a Table

The following is a discussion of the operations and functions the application requester and the application server perform. This is a brief description of some of the parameters for the DDM commands. See the DDM Reference for a detailed description of the parameters.

1 After the application requester and the application server have established the proper connection (described in Figure 4-2 on page 55), the SQL Describe Table statement command can be executed. This command requests that a description of the relational database table named in the SQLOBJNAM command data object be returned to the requester.

The application requester creates the Describe RDB Table (DSCRDBTBL) command. It places the SQL table name that is to be described in the SQLOBJNAM command data object and sends it to the application server.

2 The application server receives and processes the DSCRDBTBL command. Normal completion of this command results in the description of the named relational database table being returned in the SQLDARD reply data object. If errors occur during the execution of the command, the SQLDARD reply data object reports the exception conditions that the relational database detected.

Note: If the current unit of work had been abnormally terminated, the application server would have returned an SQLCARD reply data object and an ABNUOWRM reply message instead of the SQLDARD reply data object.

3 If the SQLDARD reply data object indicates that the DSCRDBTBL command was successful, the application requester returns the table description to the application that is attempting to execute the Describe Table SQL statement.

At this point, the application/application requester can continue with additional defined DRDA flows.

If the SQLDARD or SQLCARD reply data object that the application server returns to the application requester indicates that the DSCRDBTBL command was not successful, the application requester returns an exception to the application that is attempting to execute the Describe Table SQL statement.

4.4.11 Executing a Dynamic SQL Statement

Figure 4-21 indicates the DDM commands and replies that flow when a user is executing an SQL statement that has not been previously bound to the relational database or prepared as an SQL statement within the current unit of work. The usual result is that the application server makes the expected changes in the relational database (within the scope of the current unit of work) after the statement successfully executes.

If connection is between an application server and database server, any new or changed special register settings must be sent using the EXCSQLSET command prior to activating or processing queries. The EXCSQLSET command is recommended to be chained next SQL command.

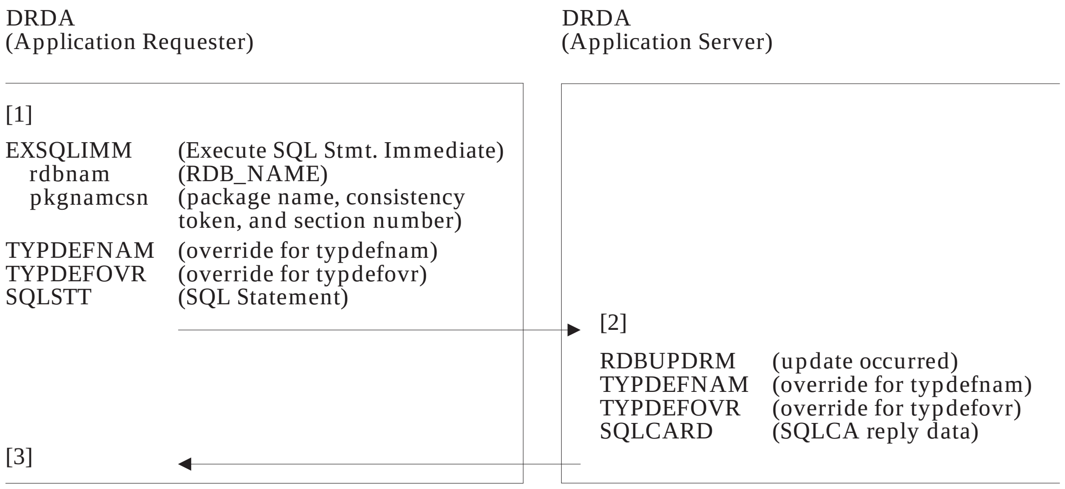

DRDA
(Application Requester)
DRDA
(Application Server)
[1]
EXSQLIMM (Execute SQL Stmt. Immediate)
rdbnam (RDB_NAME)
pkgnamcsn (package name, consistency token, and section number)
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLSTT (SQL Statement)
[2]
RDBUPDRM (update occurred)
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLCARD (SQLCA reply data)
[3]

Figure 4-21 DRDA Flows: Immediate Execution of SQL Work

The following is a discussion of the operations and functions the application requester and the application server perform. This is a brief description of some of the parameters for the DDM commands. See the DDM Reference for a detailed description of the parameters.

1 After the application requester and the application server have established the proper connection (described in Figure 4-2 on page 55), the user can execute some SQL statements without binding them in a package (see Section 4.4.3 on page 67) or preparing them (see Section 4.4.8 on page 108). These SQL statements are limited to those with no input host application variables or output row data. Other commands can precede or follow the execution of this SQL statement and be part of the same unit of work.

The application requester creates the EXECUTE IMMEDIATE SQL statement (EXCSQLIMM) command by providing the correct package name, consistency token, and section number in the *pkgnamcsn* parameter. The optional *rdbcmtok* parameter informs the RDB whether or not it can process commit and rollback operations. The SQL statement that is to be executed is placed in the SQLSTT command data and sent to the application server.

2 The application server receives and processes the EXCSQLIMM command. It executes the requested statement. The relational database reflects the results (within the scope of the unit of work), and the application server returns an SQLCARD reply data object. If errors occur during the execution of the statement, the relational database remains unchanged and the SQLCARD reply data object contains an error condition indicator.

If the executed SQL statement is either a COMMIT or ROLLBACK, then see Section 4.4.12.1 on page 117 and Section 4.4.12.2 on page 120 for a description of commit and rollback processing in DRDA.

3 If the SQLCARD reply data object that the application server returned to the application requester indicates that the EXCSQLIMM command was not successful, the application requester returns an exception to the application that is attempting to execute the SQL statement.

Assuming it has received an SQLCARD reply data object indicating normal completion, the application requester proceeds to return an indication of the normal completion to the application.

At this point, the application/application requester can continue with additional defined DRDA flows.

4.4.12 Commitment of Work in DRDA

An application program normally initiates commit or rollback processing by calling the sync point manager, or by executing a COMMIT or ROLLBACK SQL statement. Both static and dynamic SQL COMMIT and ROLLBACK requests are valid in DRDA. In addition, a commit operation can also occur as a result of an application program executing a stored procedure that has been defined with the *commit on return* attribute. DRDA, however, does not support any COMMIT or ROLLBACK options that might affect cursor positioning. In particular, cursor positioning, except for cursors with the HOLD option, is lost during commit and rollback processing in DRDA environments.

The SQL application should explicitly commit or roll back before termination. If the SQL application is using the services of the sync point manager, and it terminates normally but does not explicitly commit or rollback, the sync point manager will invoke the commit function. If the SQL application is not using the services of the sync point manager, and it terminates normally but does not explicitly commit or rollback, then the application requester must invoke the commit function. The scope of the commit includes all relational databases that were part of the unit of work as defined by SQL connection semantics. This can include local relational databases that are not using DRDA protocols but might be under application requester control. If the SQL application is not using the services of the sync point manager, and it terminates abnormally, the application requester can invoke the rollback function, and it can depend on the implicit rollback that accompanies network connection termination for databases connected using DRDA.

On unprotected network connections, the application server must inform the application requester whenever commit or rollback processing completes at the application server, except when the rollback is a result of the network connection termination. For application servers supported by protected network connections, the sync point manager informs the application requester when commit or rollback processing is complete.

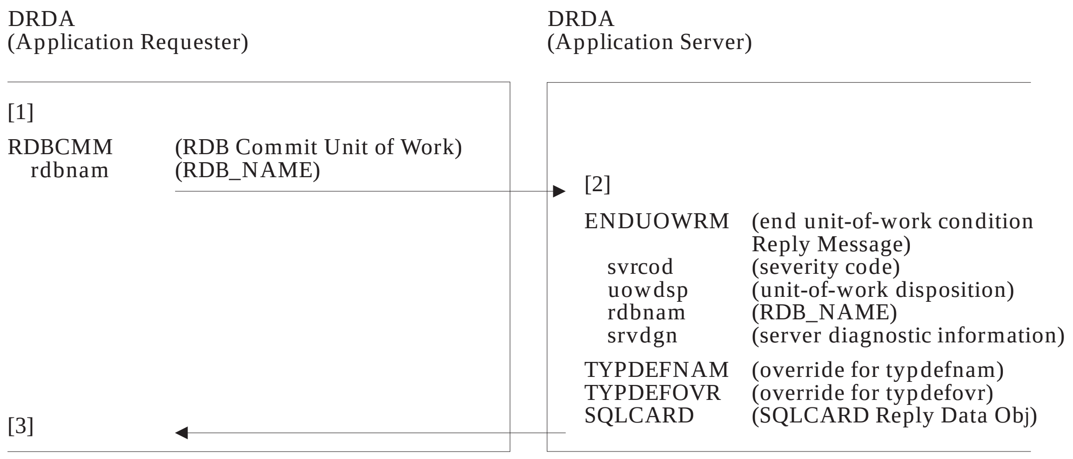
Deadlocks or abnormal ending conditions at the application server can also cause rollback processing at the application server.

Within DRDA environments, all forms of commit and rollback requests are equivalent.

In DRDA, the application requester plays an important role in helping coordinate the commitment or roll back of work at all application servers involved in the unit of work. For Remote Unit of Work, this is one application server; for Distributed Unit of Work, it can be many application servers. The application requester is responsible for interoperating with the local sync point manager, if it is involved in the unit of work. For Distributed Unit of Work, this interoperation includes coordinating the work that is not supported by two-phase commit protected network connections, and with the sync point manager that coordinates the work supported by two-phase commit protected network connections. The responsibility of the application requester also includes the proper management of the update privileges at all the application servers, so that the integrity of the unit of work can be preserved during commit processing. Also included in the commitment and rollback processing is the proper management of the network connections that support the connections to the application servers. The application requester must terminate these network connections when they are no longer needed, as defined by SQL connection semantics.

4.4.12.1 Commitment of Work in a Remote Unit of Work

Figure 4-22 indicates the DDM commands and replies that flow to commit the unit of work on DRDA Level 3 connections. Figure 4-23 on page 119 indicates the flows when commit is included in a stored procedure on a DRDA level 1 application server.



DRDA
(Application Requester)
DRDA
(Application Server)
[1]
RDBCMM (RDB Commit Unit of Work)
rdbnam (RDB_NAME)
[2]
ENDUOWRM (end unit-of-work condition Reply Message)
svrcod (severity code)
uowdsp (unit-of-work disposition)
rdbnam (RDB_NAME)
srvdgn (server diagnostic information)
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLCARD (SQLCARD Reply Data Obj)
[3]

Figure 4-22 DRDA Flows: Commit a Remote Unit of Work

The following is a discussion of the operations and functions the application requester and the application server perform. This is a brief description of some of the parameters for the DDM commands. For a detailed description of the parameters, see the DDM Reference.

1 Assuming that all required work for the application requester is complete, and the last application command is a static commit, or the application terminates normally without issuing a commit, the application requester generates a Relational Database Commit Unit of Work (RDBCMM) command.

The application requester can alternatively have created an RDB Rollback Unit of Work (RDBRLLBCK) command if the changes made during the last unit of work should not be made a permanent part of the relational database.

The application requester sends the command (in this case the RDBCMM command) to the application server.

Note: Other acceptable DRDA flows can accomplish the commit or rollback of a unit of work, but this method is preferred. However, for compatibility with existing applications, the following methods are also acceptable.

The application can use the EXECUTE IMMEDIATE flows described in Section 4.4.11 on page 114, where the SQL statement to be executed (specified in the SQLSTT command data) is *COMMIT <WORK>* or *ROLLBACK <WORK>*.

The application can use the prepare and execute flows described in Section 4.4.8 on page 108, where the SQL statement to be prepared and then executed (specified in the SQLSTT command data) is *COMMIT <WORK>* or *ROLLBACK <WORK>*.

Occurrences of *COMMIT <WORK>* or *ROLLBACK <WORK>* in the application source do not result in BNDSQLSTT commands being sent from the application requester to the application server during BIND processing.

At application execution time, the application requester sends the corresponding RDBCMM or RDBRLLBCK command when these SQL statements are to be executed.

The information enclosed in the < > is optional.

2 The application server receives and processes the RDBCMM command. If the application server finds no errors, the application server makes the remaining changes in the relational database permanent, completes the unit of work, and returns an ENDUOWRM reply message (indicating the application server completed the unit of work) and a normal SQLCARD reply data object.

- The ENDUOWRM reply message always precedes the SQLCARD reply data object when they are in response to an RDBCMM command.
- The application server returns the ENDUOWRM reply message as a result of any command that causes normal termination of a unit of work. These commands include RDBCMM, RDBRLLBCK, EXCSQLIMM (where the SQL statement being executed is either a COMMIT or ROLLBACK), and EXCSQLSTT (where the dynamically prepared SQL statement being executed is COMMIT or ROLLBACK).

Otherwise, the SQLCARD reply data object indicates a single error. The application server returns the ENDUOWRM reply message and the SQLCARD to the application requester and rolls back the unit of work.

3 The application requester:

- Receives the ENDUOWRM and SQLCARD from the application server.
- Checks the *uowdsp* parameter for the status of the unit of work (committed or rolled back).
- Resets its indication of what cursors are open.
- Returns the SQLCA to the application if the application has not terminated.

A rollback will close all cursors.

If the application has terminated, the application requester terminates the network connection to the application server using verbs and calls described in Part 3, Network Protocols.

Figure 4-23 on page 119 indicates the DDM commands and replies that flow during the execution of a statement that invokes a stored procedure such as a CALL statement that was bound by the bind process or the PRPSQLSTT command. The stored procedure referenced by the CALL performs a series of SQL statements which includes one or more requests to commit.

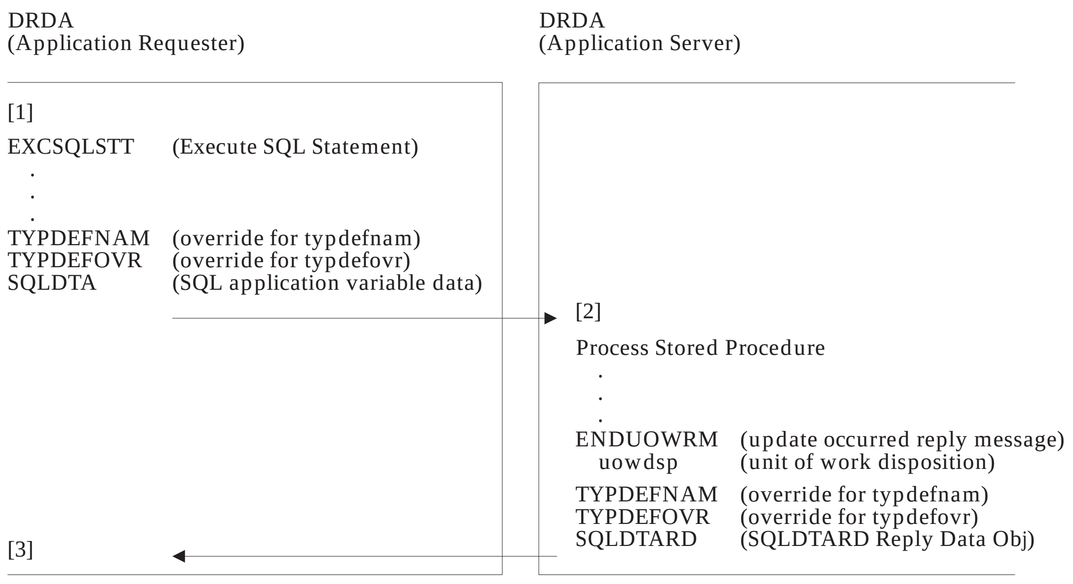

DRDA
(Application Requester)
DRDA
(Application Server)
[1]
EXCSQLSTT (Execute SQL Statement)
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLDTA (SQL application variable data)
[2]
Process Stored Procedure
ENDUOWRM (update occurred reply message)
uowdsp (unit of work disposition)
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLDTARD (SQLDTARD Reply Data Obj)
[3]

Figure 4-23 DRDA Flows: Executing a Bound SQL CALL Statement

The following is a discussion of the operations and functions the application requester and the application server perform. This document provides a brief description of some of the parameters for the DDM commands. See the DDM Reference for a detailed description of the parameters.

1 After the application requester and the application server have established proper connection, the application requester sends the command and command data to the application server. In this case, the EXCSQLSTT references a CALL statement for a stored procedure located at the application server.

2 The application server receives and processes the EXCSQLSTT command which invokes the stored procedure. In this example, the stored procedure processing includes some SQL requests to commit the unit of work. The requests to commit are processed at the application server and the stored procedure continues processing until the procedure is exited. The application server returns an ENDUOWRM with the *uowdsp* set to indicate at least one commit occurred in the stored procedure. Regardless of the number of commit or rollbacks that occur within the stored procedure, only one ENDUOWRM is returned. If a rollback occurred along with a commit, then *uowdsp* is set to indicate a rollback occurred.

If there are host variables, an SQLDTARD is returned along with the ENDUOWRM.

3 The application requester receives the results from the EXCSQLSTT statement and returns the results to the application. See Section 4.4.7 on page 96 for details.

The application requester also performs cursor management operations dependent on the value of the *uowdsp* parameter.

4.4.12.2 Commitment of Work in a Distributed Unit of Work

The following sections describe the environment where the application directs the distribution of work. The application explicitly connects to multiple databases within the same unit of work, performs operations, commits, or rollbacks, and expects all resources to commit or roll back together. It is the responsibility of the application requester to manage the connections and coordinate or participate in the coordination of the commitment or rollback of all application server participants in the unit of work.

Coexistence

To help the application requester manage the application server connections and still provide coexistence support for old applications, the application requester must have information available to it that describes whether the application is going to use resource recovery in the unit of work. For example, if the application is to update multiple resources (database and possibly non-database) per unit of work, then the application requires the services of a sync point manager to coordinate resource recovery,and the application requester must know this to aid in managing the connections to the application servers and update restrictions at the application servers. This information is the basis for defining the DRDA update rules defined later in this section. The application requester's acquisition of this application information is not defined by DRDA, but it is required to be available at the application requester.

There are two possible environments that result from the application's use of the services of a sync point manager for resource recovery. These environments are Single Relational Database Update and Multi-Relational Database Update.

The Single Relational Database Update environment is where the services of a sync point manager are not required to perform resource recovery for the unit of work. Because of this, only one resource can be updated. This resource may or may not be a database resource, but within the scope of this document, it is restricted to a database resource. All other resources are restricted to read-only.

The Multi-Relational Database Update environment is where the services of a sync point manager are required to perform resource recovery for the unit of work. Because of this, all application servers that are on network connections protected by two-phase commit protocols have update privileges. All application servers that are not on network connections protected by two-phase commit protocols are restricted to read-only.

Figure 4-24 on page 121 displays a Distributed Unit of Work application requester with connections to three application servers. AS1 is using DRDA Remote Unit of Work protocols. AS2 and AS3 are using Distributed Unit of Work protocols, but with different levels—the sync point manager (SYNCPTMGR Levels 4 and 5, respectively).

In Figure 4-24 on page 121, if the application is not using the services of the sync point manager for resource recovery in the unit of work, then either AS1, AS2, or AS3 can have update privileges, and the other two are restricted to read-only. This is an example of single relational database update. If the application is using the sync point manager for resource recovery in the unit of work, AS1 and AS2 are always restricted read-only, AS3 and any other application servers supported by two-phase commit protected network connections can have update privileges. This is an example of multi-relational database update.

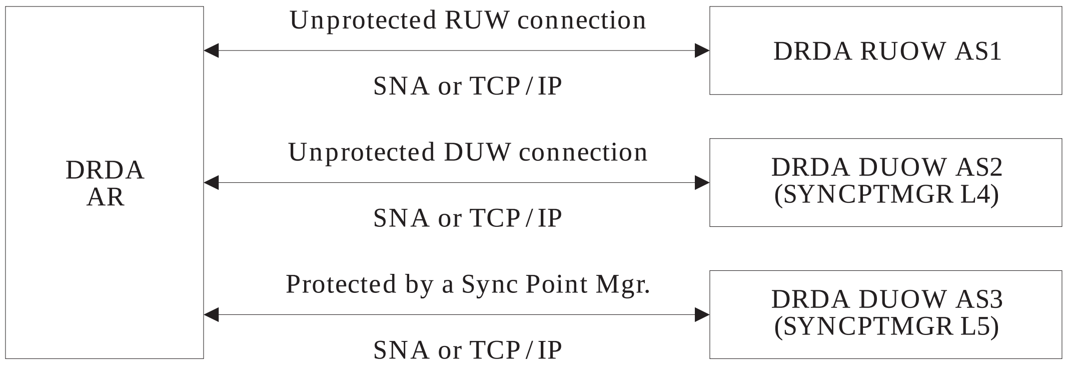


Figure 4-24 DRDA Sample Configuration

The application requester is responsible for managing the operation of the environment to make sure that any update restrictions in effect are enforced and to take the necessary steps to ensure rollback of the unit of work if any update restrictions are violated. The application requester, in cooperation with the sync point manager (if available), is also responsible for coordinating the commit or rollback of all DRDA participants in the unit of work.

The rules for deciding which application server gets update privileges and when are as follows.

Note: The rules are based on the goal that the full set of functions in SQLAM Level 5 are available, no matter what type of distribution (or sync point manager level) is supported.

- If the application is not using the services of a sync point manager in the unit of work:
  - — When connecting to an application server using Remote Unit of Work protocols, the application server is allowed updates if only:
    - — There are no existing connections to any other application servers.
    - — All existing connections are to application servers using Remote Unit of Work protocols, and these application servers are restricted to read-only.
  - — If a connection exists to an application server using Remote Unit of Work protocols with update privileges, all other application servers are restricted to read-only. Otherwise, for the duration of any single unit of work, the first application server using Distributed Unit of Work protocols that performs an update is given update privileges, and all other application servers are restricted to read-only.
- If the application is using the services of a sync point manager for the unit of work, only connections to application servers using Distributed Unit of Work protocols that are supported by two-phase commit protected network connections are allowed update privileges.

The application requester uses the RDBALWUPD parameter on ACCRDB as defined in rule CR6 to control the update, dynamic COMMIT, and dynamic ROLLBACK privileges on application servers.

For Distributed Unit of Work application servers, the application requester is notified by the application servers the first time a DDM command results in an update at the application server within the unit of work. This information is passed to the application requester on the DDM reply message RDBUPDRM. Figure 4-25 on page 122 is an example of this flow for EXCSQLIMM.

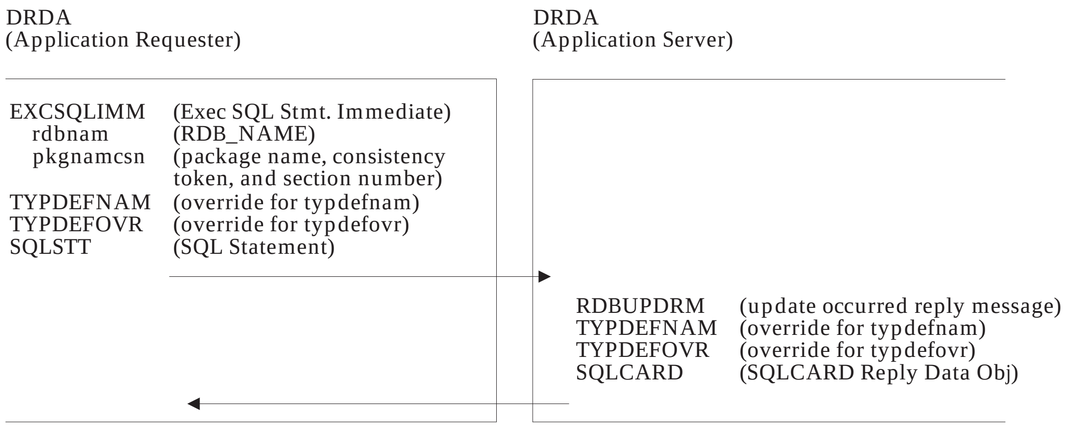

DRDA
(Application Requester)
DRDA
(Application Server)
EXCSQLIMM (Exec SQL Stmt. Immediate)
rdbnam (RDB_NAME)
pkgnamcsn (package name, consistency token, and section number)
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLSTT (SQL Statement)
RDBUPDRM (update occurred reply message)
TYPDEFNAM (override for typdefnam)
TYPDEFOVR (override for typdefovr)
SQLCARD (SQLCARD Reply Data Obj)

Figure 4-25 DRDA RDBUPDRM Example Flow

When the application requester receives the RDBUPDRM, it checks whether this application server is allowed updates. If not, the application requester must ensure that the unit of work rolls back.

An application server can return an RDBUPDRM after every update, but it is required only after the first update.

Commit and Rollback Scenarios

This section provides several scenarios to show the steps for committing and rolling back a logical unit of work. The scenarios are categorized by configurations. The configurations are different in terms of single relational database update using Remote Unit of Work protocols at an application server, single relational database update using Distributed Unit of Work protocols at an application server, and multi-relational database update. The single relational database update scenarios are by definition not working under sync point management control for resource recovery. The multi-relational database update scenarios are, by definition, working under sync point management control for resource recovery.

In the scenarios, the steps for dynamic commit requests, dynamic rollback requests or execution requests of stored procedures defined with the *commit on return* attribute assume the request is directed to an application server that is allowed updates. If the request is directed to a read-only (as a result of *rdbalwupd* on ACCRDB) restricted application server operating in a Remote Unit of Work environment introduced in DRDA Level 1, an SQLSTATE of X‘2D528’ for commit or SQLSTATE X‘2D529’ for rollback is returned to the application requester. If the local environment allows it, the application requester should initiate processing of commit or rollback based on the SQLSTATE. If the local environment does not allow the application requester to initiate commit or rollback, the SQLSTATE should be returned to the application.

If a commit or rollback request is application-directed to a read-only application server operating in a Distributed Unit of Work environment, a DDM reply message CMMRQSRM, with the *cmmtyp* parameter indicating a commit or rollback, is returned to the application requester. If the local environment allows it, the application requester will initiate commit or rollback processing based on the value in the *cmmtyp* parameter. If the local environment does not allow the application requester to initiate commit or rollback, an SQLCA should be returned to the application with SQLSTATE X‘2D528’ enclosed for commit or SQLSTATE X‘2D529’ enclosed for rollback.

An application server only begins commit processing if it is requested to commit. When using the communications sync point manager, if an application requester receives a request to commit (for example, an LU 6.2 TAKE_SYNCPT or DDM SYNCCTL request to commit command) on a network connection with an application server, the application requester must ensure that a rollback occurs.

Single RDB Update When Using Remote Unit of Work

In the following commit and rollback scenario, the application is not using the services of the sync point manager to coordinate resource recovery for the unit of work. The application server that is allowed updates is operating at DRDA Remote Unit of Work (see AS1 in Figure 4-26) on an unprotected network connection.

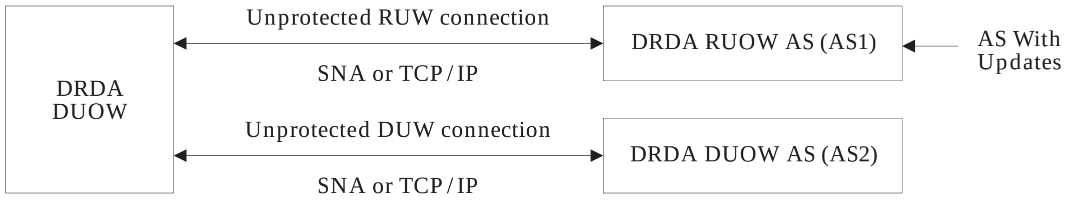


Figure 4-26 Single RDB Update at a DRDA Remote Unit of Work AS

All other application servers are restricted to read-only and, for this scenario, are assumed to be on unprotected network connections. The scenario only describes the commit and rollback flows. The application requester is responsible for performing all other local processing that is required to complete the commit or rollback at the application requester.

- Dynamic Commit Steps

  1. The commit request passes to the application server that is allowed updates using either the EXCSQLIMM command or the EXCSQLSTT command. The EXCSQLIMM command flow is described in Figure 4-21 on page 114. The EXCSQLSTT command flow is described in Figure 4-15 on page 97.
  2. The update application server is operating at DRDA Remote Unit of Work, so commit processing occurs at the application server. The application server returns an ENDUOWRM and SQLCARD to the application requester with the *uowdsp* parameter on the ENDUOWRM indicating a commit succeeded at the application server.
  3. The application requester receives the ENDUOWRM and SQLCARD from the update application server. The application requester checks the value in the *uowdsp* parameter and sends an RDBCMM command to all read-only application servers. See Figure 4-22 on page 117 for a description of the command flows for RDBCMM.
  4. The read-only application servers receive the RDBCMM command and perform the commit. The application servers return the results of the commits using ENDUOWRMs and SQLCARDs.
  5. The application requester receives the results from the read-only application servers.

     Regardless of the outcome from the read-only application servers, the result returned to the application must reflect the status of the work at the update application server.

     If a read-only application server rolls back when it is asked to commit, the next time the application performs a request to any application server, the application requester returns SQLSTATE X‘51021’ to the application to inform it that it must issue a static rollback. The application requester does not need to return an SQLSTATE X‘51021’ if the application requester performed an implicit rollback and informed the application the commit was successful and an implicit rollback occurred.

- Dynamic Rollback Steps

  1. The rollback request passes to the application server that is allowed updates using either the EXCSQLIMM command or the EXCSQLSTT command. The EXCSQLIMM command flow is described in Figure 4-21 on page 114. The EXCSQLSTT command flow is described in Figure 4-15 on page 97.

  2. The update application server is operating at DRDA Remote Unit of Work, so rollback processing occurs at the application server. The application server returns an ENDUOWRM and SQLCARD to the application requester with the *uowdsp* parameter on the ENDUOWRM indicating the rollback succeeded at the application server.

  3. The application requester receives the ENDUOWRM and SQLCARD from the update application server. The application requester checks the value in the *uowdsp* parameter and sends an RDBRLLBCK command to all read-only application servers.

  4. The read-only application servers receive the RDBRLLBCK command and perform the rollback. The application servers return the results of the rollbacks using ENDUOWRMs and SQLCARDs.

  5. The application requester receives the results from the read-only application servers and returns to the application the status of the work at the update application server.

     Because the unit of work rolled back, the application requester resets all cursors to a closed state.

- Static Commit Steps

  1. The application requester receives the request for the embedded commit.

     If the local environment does not allow static commits, the application requester must return to the application an SQLCA with an SQLSTATE value of X‘2D521’.

  2. The application requester sends an RDBCMM command to all read-only application servers.

  3. The read-only application servers receive the RDBCMM command and perform the commit. The application servers return the results of the commits using ENDUOWRMs and SQLCARDs.

  4. The application requester receives the results from the read-only application servers. If all read-only application servers commit successfully, the application requester sends an RDBCMM command to the application server that is allowed updates.

     If a read-only application server rolls back when it is asked to commit, the application requester sends an RDBRLLBCK command to the application server that performed the update. The application requester also rolls back the read-only application servers by sending an RDBRLLBCK command to the application servers.

  5. The application server that performed the update receives the RDBCMM command and performs the commit. The application server returns the result of the commit using an ENDUOWRM and SQLCARD.

  6. The application requester receives the result from the update application server and returns the status of the work at the update application server to the application.

     If the update application server rolls back when it is asked to commit, the application requester rolls back the read-only application servers by sending an RDBRLLBCK command to the application servers.

If the unit of work rolled back, the application requester resets all cursors to a closed state.

- Static Rollback Steps

  1. The application requester receives the request for the embedded rollback.

     If the local environment does not allow static commits, the application requester must return to the application an SQLCA with an SQLSTATE value of X‘2D521’.

  2. The application requester sends an RDBRLLBCK command to all application servers.

  3. The application servers receive the RDBRLLBCK command and perform the rollback. The application servers return the results of the rollbacks using ENDUOWRMs and SQLCARDs.

  4. The application requester receives the results from the application servers and returns to the application the status of the work at the update application server.

     Because the unit of work rolled back, the application requester resets all cursors to a closed state.

- Commit Steps

  (for a stored procedure defined with the *commit on return* attribute)

  1. The application server initiates commit processing when a stored procedure defined with the *commit on return* attribute terminates.

  2. The update application server is operating at DRDA Remote Unit of Work, so commit processing occurs at the application server. The application server returns an ENDUOWRM and either an SQLCARD or SQLDTARD to the application requester with the *uowdsp* parameter on the ENDUOWRM indicating a commit succeeded at the application server.

  3. The application requester receives the ENDUOWRM and the SQLCARD or SQLDTARD from the update application server. The application requester checks the value in the *uowdsp* parameter and sends an RDBCMM command to all read-only application servers.

  4. The read-only application servers receive the RDBCMM command and perform the commit. The application servers return the results of the commits using ENDUOWRMs and SQLCARDs.

  5. The application requester receives the results from the read-only application servers.

     Regardless of the outcome from the read-only application servers, the result returned to the application must reflect the status of the work at the update application server.

     If a read-only application server rolls back when it is asked to commit, the next time the application performs a request to any application server, the application requester returns SQLSTATE X‘51021’ to the application to inform it that it must issue a static rollback. The application requester does not need to return an SQLSTATE X‘51021’ if the application requester performed an implicit rollback and informed the application the commit was successful and an implicit rollback occurred.

Single RDB Update Using Distributed Unit of Work

In this commit and rollback scenario, the application is not using the services of the sync point manager to coordinate resource recovery for the unit of work. The application server that is allowed updates is operating using Distributed Unit of Work (see AS2 in Figure 4-27). AS1 is operating using Remote Unit of Work, and is restricted to read-only. AS1 is restricted to an unprotected network connection. For this scenario, AS2 is on an unprotected network connection. The scenario describes only the commit and rollback flows. The application requester is responsible for performing all other local processing that is required to complete the commit or rollback at the application requester. There are slightly different scenarios depending on whether the parameter *rdbcmtok* has the value TRUE.

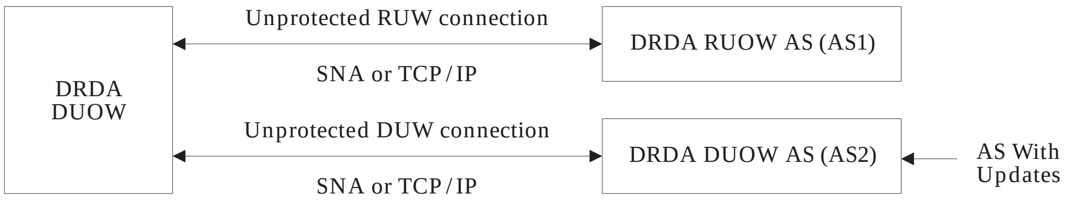


Figure 4-27 Single RDB Update Using Distributed Unit of Work

- Dynamic Commit Steps

  (*rdbcmtok* value is FALSE)

  1. The commit request passes to the application server that is allowed updates using either the EXCSQLIMM command or the EXCSQLSTT command. The EXCSQLIMM command flow is described in Figure 4-21 on page 114. The EXCSQLSTT command flow is described in Figure 4-15 on page 97.
  2. Dynamic commits are not processed at application servers in this situation, so the application server sends a CMMRQSRM with the value of the *cmmtyp* parameter set to commit.
  3. The application requester receives the CMMRQSRM, checks the value in the *cmmtyp* parameter and sends an RDBCMM command to all read-only application servers.

     The local environment can require the results of a failed dynamic commit to be returned to the application instead of continuing with the commit processing. In this case, the application requester returns to the application an SQLCA with an SQLSTATE value of X‘2D528’.
  4. The read-only application servers receive the RDBCMM command and perform the commit. The application servers return the results of the commits using an ENDUOWRM and SQLCARDs.
  5. The application requester receives the results from the read-only application servers. If all read-only application servers commit successfully, the application requester sends an RDBCMM command to the application server that is allowed updates.

     If a read-only application server rolls back when it is asked to commit, the application requester sends an RDBRLLBCK to the update application server. The application requester also rolls back the read-only application servers by sending an RDBRLLBCK command to those application servers.
  6. The update application server receives the RDBCMM command and performs the commit. The application server returns the result of the commit using an ENDUOWRM

and SQLCARD.

7. The application requester receives the result from the update application server and returns the status of the work at the update application server to the application.

   If the update application server rolls back when it is asked to commit, the application requester rolls back the read-only application servers by sending an RDBRLLBCK command to those application servers.

   If the unit of work rolled back, the application requester resets all cursors to a closed state.

- Dynamic Commit Steps

  (*rdbcmtok* value is TRUE)

  1. The commit request passes to the application server that is allowed updates using either the EXCSQLIMM command or the EXCSQLSTT command.
  2. Since *rdbcmtok* was specified as TRUE in the command, the application server processes the commit request and sends an ENDUOWRM with *uowdsp* set to committed and an SQLCARD. RDBUPDRM may also have to be sent.
  3. The application requester receives the ENDUOWRM and SQLCARD. In a fashion similar to a DRDA Distributed Unit of Work application requester that receives ENDUOWRM from a DRDA Remote Unit of Work application server, the application requester sends RDBCMM to all other read-only servers.
  4. The read-only application servers receive the RDBCMM command and perform the commit. The application servers return the results of the commits using ENDUOWRMs and SQLCARDs.
  5. The application requester receives the results from the read-only application servers.

     Regardless of the outcome from the read-only application servers, the result returned to the application must reflect the status of the work at the update application server.

     If a read-only application server rolls back when it is asked to commit, the next time the application performs a request to any application server, the application requester returns SQLSTATE X‘51021’ to the application to inform it that it must issue a static rollback. The application requester does not need to return an SQLSTATE X‘51021’ if the application requester performed an implicit rollback and informed the application the commit was successful and an implicit rollback occurred.

- Dynamic Rollback Steps

  (*rdbcmtok* value of FALSE)

  1. The rollback request passes to the application server that is allowed updates using either the EXCSQLIMM command or the EXCSQLSTT command. The EXCSQLIMM command flow is described in Figure 4-21 on page 114. The EXCSQLSTT command flow is described in Figure 4-15 on page 97. The application server sends a CMMRQSRM with the value of the *cmmtyp* parameter set to rollback.
  2. The application requester receives CMMRQSRM and then sends an RDBRLLBCK command to all application servers.

     The local environment can require the results of the failed dynamic rollback to be returned to the application instead of continuing with the rollback processing. The application requester returns to the application an SQLCA with an SQLSTATE value of X‘2D529’.

3. The application servers receive the RDBRLLBCK command and perform the rollback. The application servers return the results of the rollbacks using ENDUOWRMs and SQLCARDs.
4. The application requester receives the results from the application servers and returns to the application the status of the work at the update application server.

   Because the unit of work rolled back, the application requester resets all cursors to a closed state.

- Dynamic Rollback Steps

  (*rdbcmtok* value of TRUE)

  1. The rollback request passes to the application server that is allowed updates using either the EXCSQLIMM command or the EXCSQLSTT command.
  2. Since *rdbcmtok* was specified as TRUE, the RDB processes the rollback and sends ENDUOWRM with the value of *uowdsp* set to rolled back and an SQLCARD to the application requester.
  3. The application requester receives the ENDUOWRM and the SQLCARD. The application requester then sends RDBRLLBCK to all the other servers.
  4. The application servers receive the RDBRLLBCK command and perform the rollback. The application servers return the results of the rollbacks using ENDUOWRMs and SQLCARDs.
  5. The application requester receives the results from the application servers and returns to the application the status of the work at the update application server.

     Because the unit of work rolled back, the application requester resets all cursors to a closed state.

- Commit Steps

  (for a Stored Procedure defined with the *commit on return* attribute, *rdbcmtok* value is FALSE)

  1. The application server initiates commit processing when the stored procedure defined with the *commit on return* attribute terminates.
  2. Commits are not processed at the application server in this situation, so the application server sends a CMMRQSRM with the value of the *cmmtyp* parameter set to commit.
  3. The remaining steps are the same as steps 3 through 7 of the dynamic commit (*rdbcmtok* value is FALSE) scenario above.

- Commit Steps

  (for a Stored Procedure defined with the *commit on return* attribute, *rdbcmtok* value is TRUE)

  1. The application server initiates commit processing when the stored procedure defined with the *commit on return* attribute terminates.
  2. Since *rdbcmtok* was specified as TRUE in the command, the application server processes the commit request and sends an ENDUOWRM with *uowdsp* set to committed and either an SQLCARD or SQLDTARD. Note that RDBUPDRM may also have to be sent.
  3. The application requester receives the ENDUOWRM and the SQLCARD or SQLDTARD. In a fashion similar to a DRDA Distributed Unit of Work application requester that receives ENDUOWRM from a DRDA Remote Unit of Work application server, the application requester sends RDBCMM to all other read-only servers.

4. The read-only application servers receive the RDBCMM command and perform the commit. The application servers return the results of the commits using ENDUOWRMs and SQLCARDs.

5. The application requester receives the results from the read-only application servers.

   Regardless of the outcome from the read-only application servers, the result returned to the application must reflect the status of the work at the update application server.

   If a read-only application server rolls back when it is asked to commit, the next time the application performs a request to any application server, the application requester returns SQLSTATE X‘51021’ to the application to inform it that it must issue a static rollback. The application requester does not need to return an SQLSTATE X‘51021’ if the application requester performed an implicit rollback and informed the application the commit was successful and an implicit rollback occurred.

- Static Commit Steps

  1. The application requester receives the request for the embedded commit.

     If the local environment does not allow static commits, the application requester must return to the application an SQLCA with an SQLSTATE value of X‘2D521’.

  2. The application requester sends an RDBCMM command to all read-only application servers.

  3. The rest of the steps are identical to steps 4 through 7 for the dynamic commit steps in this scenario.

- Static Rollback Steps

  1. The application requester receives the request for the embedded commit rollback.

     If the local environment does not allow static commits, the application requester must return to the application an SQLCA with an SQLSTATE value of X‘2D521’.

  2. The application requester sends an RDBRLLBCK command to all application servers.

  3. The rest of the steps are identical to steps 4 through 7 for the dynamic rollback steps in this scenario.

Multi-RDB Update

In this commit and rollback scenario, the application uses the services of the sync point manager to coordinate resource recovery for the unit of work. All application servers using protected Distributed Unit of Work connections are allowed updates (see AS3 in Figure 4-28). AS1 is operating using Remote Unit of Work. AS2 is operating using Distributed Unit of Work but not protected by a sync point manager. AS1 and AS2 are restricted to read-only. This scenario describes only the commit and rollback flows. The application requester is responsible for all other local processing that is required to complete the commit or rollback at the application requester.

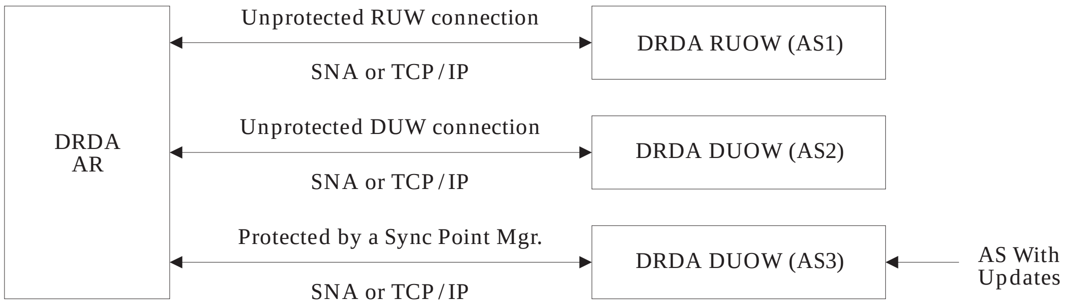


Figure 4-28 Multi-Relational Database Update

When the application requester calls the sync point manager on behalf of the application, the local sync point manager interface is being used. This scenario assumes a Resource Recovery Manager interface is being used or the DDM sync point manager Level 5 was identified on the DDM EXCSAT command, although a private interface is also valid. For example, assume the Resource Recovery Manager calls SRRCMIT and SRRBACK, which are examples of a sync point manager interface for COMMIT and ROLLBACK. When the application requester participates as a resource manager, the syntax for the sync point manager interface is not described because it is specific to the operating environment.

- Dynamic Commit Steps

  1. The commit request passes to one of the application servers that is allowed updates using either the EXCSQLIMM command or the EXCSQLSTT command. The EXCSQLIMM command flow is described in Figure 4-21 on page 114. The EXCSQLSTT command flow is described in Figure 4-15 on page 97.

  2. Dynamic commits are not allowed at application servers operating at DRDA Distributed Unit of Work, so the application server sends a CMMRQSRM with the value of the *cmmtyp* parameter set to commit.

     Note that DRDA rules do not allow *rdbcmtok* to be sent to a server that is using a sync point manager.

  3. The application requester receives CMMRQSRM, checks the value in the *cmmtyp* parameter, and acting on behalf of the application, calls the Resource Recovery interface with SRRCMIT to commit all application servers that are allowed updates. This initiates the sync-point flows as described in Part 3, Network Protocols.

     The local environment can require the results of the failed dynamic commit to be returned to the application instead of continuing with the commit processing. The application requester returns to the application an SQLCA with an SQLSTATE value of X‘2D528’.

4. Because the application requester is registered with the sync point manager, the sync point manager contacts the application requester to participate in the resource recovery process.

5. When contacted during phase one of the two-phase commit process, the application requester sends an RDBCMM command to all read-only application servers.

6. The read-only application servers receive the RDBCMM command and perform the commit. The application servers return the results of the commits using ENDUOWRMs and SQLCARDs. See Part 3, Network Protocols for a discussion of the levels of sync point managers required to support update servers on protected network connections.

   If an application server is on a protected network connection, and it receives an RDBCMM command, the RDBCMM command is rejected. The application server generates an alert and returns a CMDVLTRM to the application requester.

7. The application requester receives the results from the read-only application servers. If all read-only application servers commit successfully, the application requester responds to the sync point manager interface in phase one to commit.

   If a read-only application server rolls back when it is asked to commit, or the application requester receives a CMDVLTRM from an application server, or the application requester receives any error reply message that does not allow the application requester to proceed, the application requester responds with a rollback to the sync point manager interface. The application requester also rolls back the read-only application servers by sending an RDBRLLBCK command to the application servers.

8. The sync point manager completes the resource recovery process, which includes another call to the application requester during phase two of the two-phase commit protocols.

   If the phase two call from the sync point manager is rollback, the application requester sends an RDBRLLBCK to the read-only application servers.

9. The application requester, acting on behalf of the application, receives the response from the call to the sync point manager, and returns the result of the commit to the application.

   If the unit of work rolled back, the application requester resets all cursors to a closed state.

- Dynamic Rollback Steps

  1. The rollback request tells one of the application servers that is allowed updates using either the EXCSQLIMM command or the EXCSQLSTT command. The EXCSQLIMM command flow is described in Figure 4-21 on page 114. The EXCSQLSTT command flow is described in Figure 4-15 on page 97.

  2. Dynamic rollbacks are not allowed at application servers operating at DRDA Distributed Unit of Work, so the application server sends a CMMRQSRM with the value of the *cmmtyp* parameter set to rollback.

  3. The application requester receives CMMRQSRM, checks the value in the *cmmtyp* parameter, and acting on behalf of the application, calls the Resource Recovery interface with SRRBACK to roll back all application servers that are allowed updates. This initiates the rollback flows as described in Part 3, Network Protocols.

The local environment can require the results of the failed dynamic rollback to be returned to the application instead of continuing with the rollback processing. The application requester returns to the application an SQLCA with an SQLSTATE value of X‘2D529’.

4. Because the application requester is registered with the sync point manager, the sync point manager contacts the application requester to participate in the resource recovery process.
5. When contacted during phase one of the two-phase commit process, the application requester sends an RDBRLLBCK command to all read-only application servers.
6. The read-only application servers receive the RDBRLLBCK command and perform the rollback. The application servers return the results of the rollbacks using ENDUOWRMs and SQLCARDs.

   If an application server is on a protected network connection and it receives an RDBRLLBCK command, the RDBRLLBCK command is rejected. The application server generates an alert and returns a CMDVLTRM to the application requester.
7. The application requester receives the results from the read-only application servers and replies to the sync point manager with an acknowledgement to roll back.
8. The sync point manager completes the resource recovery process and returns the results to the application requester.
9. The application requester, acting on behalf of the application, receives the response from the call to the Resource Recovery interface, and returns the result of the rollback to the application.

   Because the unit of work was rolled back, the application requester resets all cursors to a closed state.

- Commit Steps for a Stored Procedure defined with the *commit on return* attribute
  1. The application servers initiate commit processing when the stored procedure defined with the *commit on return* attribute terminates.
  2. Commits are not processed at the application server in this situation, so the application server sends a CMMRQSRM with the value of the *cmmtyp* parameter set to commit.
  3. The rest of the steps are identical to steps 3 through 9 under dynamic commit for this scenario.
- Static Commit Steps
  1. The application requester receives the request for the embedded commit.

     If the local environment does not allow static commits, the application requester must return to the application an SQLCA with an SQLSTATE of X‘2D521’.
  2. The application requester, acting on behalf of the application, calls the Resource Recovery interface or the DDM sync point manager, to commit all application servers that are allowed updates.
  3. The rest of the steps are identical to steps 4 through 9 under dynamic commit for this scenario.
- Static Rollback Steps
  1. The application requester receives the request for the embedded rollback.

If the local environment does not allow static commits, the application requester must return to the application an SQLCA with an SQLSTATE of X'2D521'.

2. The application requester, acting on behalf of the application, calls the DDM sync point manager, to roll back all application servers that are allowed updates.
3. The rest of the steps are identical to steps 4 through 9 under dynamic rollback for this scenario.

- Sync Point Manager Originating Commit Steps
  1. The application commits the unit of work by calling the DDM sync point manager.
  2. The rest of the steps are identical to steps 4 through 8 under dynamic commit for this scenario.
  3. The DDM sync point manager returns the results to the application.

     If the unit of work rolled back, the application requester resets all cursors to a closed state.
- Sync Point Manager Originating Rollback Steps
  1. The application rolls back the unit of work by calling the DDM sync point manager.
  2. The rest of the steps are identical to steps 4 through 8 under dynamic rollback for this scenario.
  3. The DDM sync point manager returns the results to the application.

     Because the unit of work rolled back, the application requester resets all cursors to a closed state.

Post-Commit Processing

After the commit processing completes as defined in **Commit and Rollback Scenarios** on page 122, the application can continue using the existing connections, and/or new ones. The connections that remain active for the new unit of work are defined by the SQL semantics. These semantics are defined by *ISO/IEC 9075: 1992, Database Language SQL.*

The update privileges for the existing connections and any new connections for the new unit of work follow the rules defined in **Coexistence** on page 120.

RDB-Initiated Rollback

In the following scenarios, a relational database has initiated rollback. A relational database initiated rollback is due to an uncontrollable event on the relational database that requires it to roll back immediately. The following scenarios describe the steps that occur, depending on which application server initiates the rollback. Figure 4-29 and Figure 4-30 show unprotected network connections between the application requester and application servers.

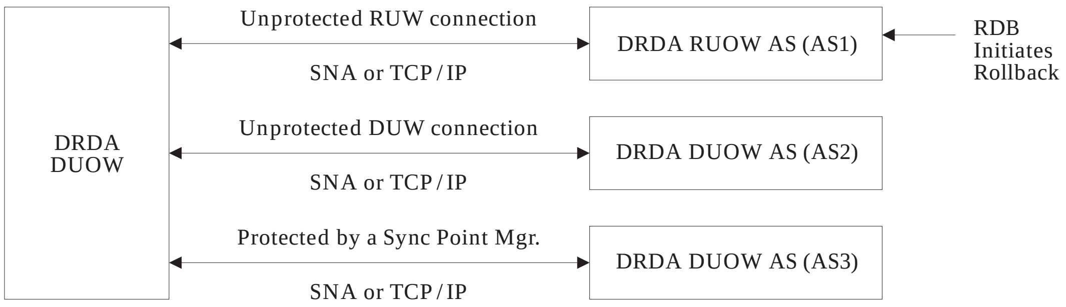


Figure 4-29 RDB at AS1 Initiates Rollback

The network connection between the application requester and AS1 is unprotected, so there is not a sync point manager involved at AS1.

1. The relational database at AS1 rolls back.
2. AS1 returns an ABNUOWRM and SQLCARD to the application requester.
3. The application requester initiates rollback processing to all other application servers involved in the unit of work. The steps for rolling back the other application servers are described in **Commit and Rollback Scenarios** on page 122. The application requester returns to the application the SQLCA received from AS1.

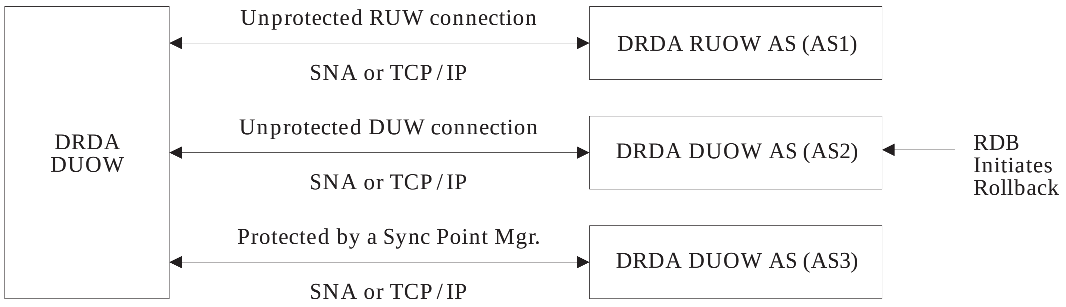


Figure 4-30 RDB at AS2 Initiates Rollback

The network connection between the application requester and AS2 is unprotected, so there is not a sync point manager involved at AS2. The steps for this scenario are the same as the previous scenario for rollback at AS1.

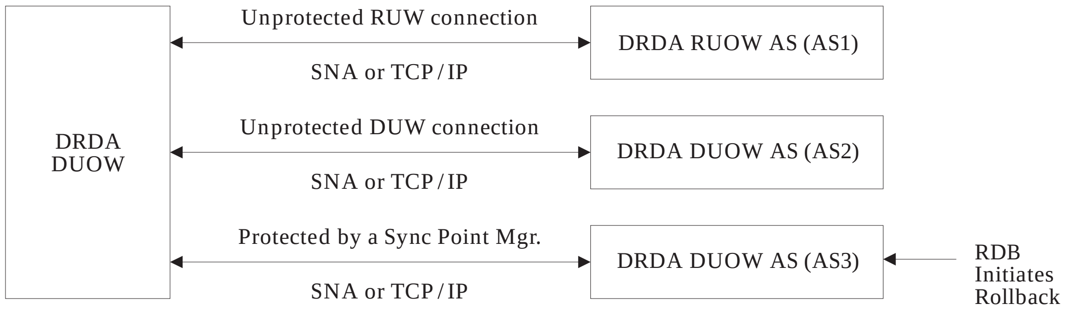


Figure 4-31 RDB at AS3 Initiates Rollback

The network connection between the application requester and AS3 is protected, so there is a sync point manager involved at AS3. This scenario is dependent on whether the sync point manager at the application server is informed to roll back before responding to the application requester. If the sync point manager is not informed to roll back at the application server before responding to the application requester, the steps are the same as the previous scenario for rollback at AS1. Replace AS1 in the steps with AS3. If the sync point manager is informed at the application server before responding to the application requester, then the following steps are in effect.

1. The sync point manager at the application server is invoked to roll back the unit of work. The process of invoking the sync point manager is dependent on the operating environment.

2. The sync point manager drives rollback, which includes sending the DDM SYNCCRD rollback reply or an LU 6.2 BACKOUT to the application requester. AS3 does not have the opportunity to send back an SQLCARD.

   Informing the application server that a rollback has occurred depends on the operating system and application server implementation. For example, if the SQLAM implementation registers itself as a protected resource manager, it will be a participant in the rollback processing.

3. The application requester receives a DDM SYNCCRD rollback reply or a TAKE_BACKOUT on the connection to the application server. Assuming the use of a Resource Recovery interface, the application requester issues a ROLLBACK request (SRRBACK) and also rolls back all other application servers. If using the DDM sync point manager, it sends the DDM SYNCCTL rollback command to roll back each of the other application servers. The steps for rolling back the other application servers are described in **Commit and Rollback Scenarios** on page 122. The application requester returns to the application an SQLCA with an SQLSTATE of X‘40504’.

   The local environment can require the application requester to respond back to the application instead of issuing the SRRBACK command. For this case, the application requester returns to the application an SQLCA with an SQLSTATE of X‘51021’.

Chapter 5

Data Definition and Exchange

The DRDA environment has several objectives for data description and data transmission. Principal among these objectives are:

- Providing a faithful representation of SQL data
- Minimizing or eliminating data conversion activity between compatible environments
- Minimizing communications traffic
- Allowing staged implementation of DRDA

5.1 Use of FD:OCA

Formatted Data Object Content Architecture (FD:OCA) is the architecture for handling exchange and interchange of field formatted information. The SQL Application Programming Interface (API) shields application programmers from the actual underlying descriptive architecture. FD:OCA provides means to describe both numeric and character information.

DRDA provides flexibility in the transmitted format of data. For example, when two identical systems are using data, no conversions should be necessary. However, when they are different, they must use clearly understood formats.

With FD:OCA, the descriptions can be sent along with the data, can be sent as a separate object before the data is sent, or can be cached for use much later. DRDA uses only a subset of the total FD:OCA function as defined in the FD:OCA documentation. Furthermore, DRDA imposes restrictions on FD:OCA as described in this chapter.

FD:OCA allows specification of Simple Data Arrays with an arbitrary number of dimensions; DRDA uses them to define only scalars. Similarly, Row LayOut (RLO) can be used repetitively to produce an arbitrarily complex structure; for DRDA RLO usage is restricted to produce two dimensional tables as the most complex data aggregate.

For more information on FD:OCA and other major terms in this chapter, see **Referenced Documents** on page xxiv. These references are also useful for background reading.

5.2 Use of Base and Option Sets

DRDA uses a subset of the descriptive architecture that FD:OCA provides. DRDA uses the following FD:OCA triplets[34] or their abbreviations in describing data.

MDD	Meta Data Definition
SDA	Simple Data Array
GDA	Group Data Array
CPT	Continue Preceding Triplet
RLO	Row LayOut

The following sections illustrate their usage.

To begin this discussion, it is important to see how data is described and presented applying DRDA concepts in the use of the FD:OCA architecture.

5.2.1 Basic FD:OCA Object Contained in DDM

Figure 5-1 is the Basic FD:OCA Object:

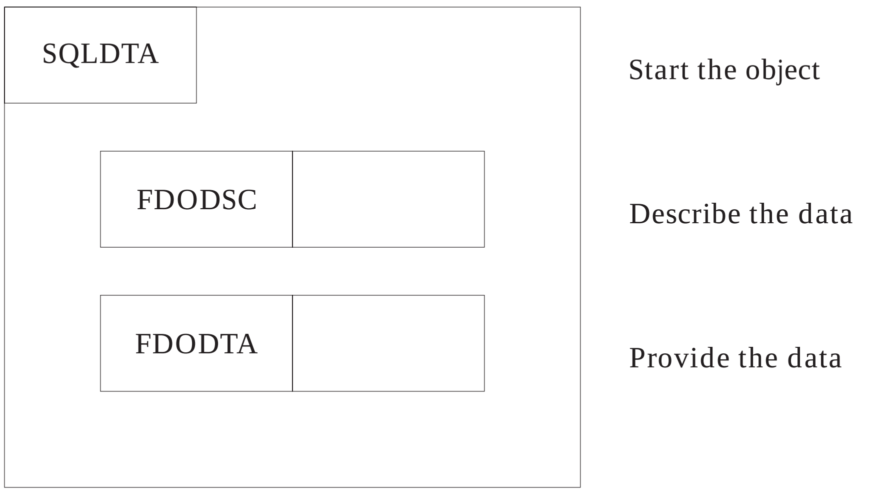


Figure 5-1 Basic FD:OCA Object

DDM defines the terms FDODSC (FD:OCA Descriptor), QRYDSC (Query Descriptor), FDODTA (FD:OCA Data), and QRYDTA (Query Data). Both descriptor objects are carriers for descriptors. Both data objects are carriers for data. In commands where the descriptor and the data are

34. Triplet is a word used in FD:OCA almost the same way the term is used in DDM. A triplet consists of three parts:
    1. A length
    2. A type
    3. The rest

    Triplets are referred to by their type, such as the RLO or Row LayOut triplet.

available simultaneously (as is the case for command data flowing to the relational database) the DDM command has a command data object (such as SQLDTA) that contains both the FDODSC and the FDODTA objects. Where the presentation of the descriptor and data can be separated in time and supplied with different commands (as is the case for query processing with OPNQRY and CNTQRY), the QRYDSC and the QRYDTA objects are used separately without an outer object. In both cases, the descriptor in the FDODSC, or QRYDSC describes the data contained in the following FDODTA object, or QRYDTA objects.

In addition, DDM Level 6 adds new terms: EXTDTA and OUTOVR. EXTDTA is a data object that allows data to flow as base data in an FDODTA or QRYDTA object or as externalized data in a separate object. If a data item is to flow as externalized data, the descriptor object contains a descriptor for the item with the FD:OCA placeholder indicator flag set on and the data object contains a FD:OCA placeholder for the data item instead of the actual data. OUTOVR is a descriptor object that allows the application requester to control the format of output host variables returned by the application server. The descriptor flows from the application requester with the command. It describes completely the data to be returned by the application server, including GDAs, SDAs, and override LEDs and MDDs as required. As with QRYDSC objects which flow separately from the corresponding QRYDTAs containing the data they describe, there is no outer object for an OUTOVR object. See the DDM Reference (Architecture Level 6) for details.

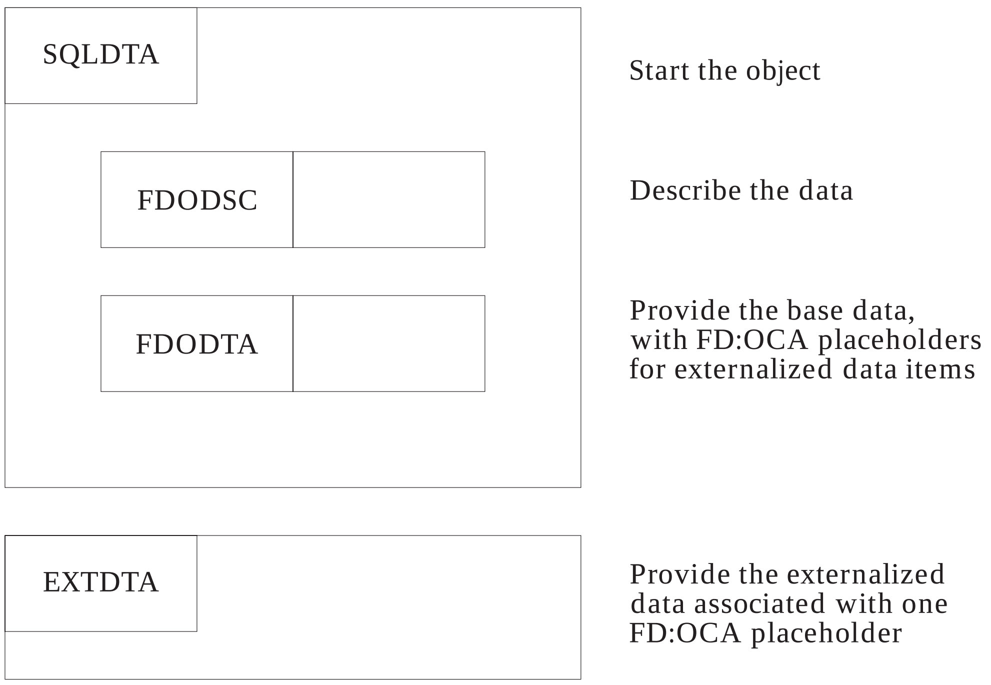

SQLDTA
Start the object
FDODSC
Describe the data
FDODTA
Provide the base data, with FD:OCA placeholders for externalized data items
EXTDTA
Provide the externalized data associated with one FD:OCA placeholder

Figure 5-2 Basic FD:OCA Object (DDM Level 6)

5.2.2 DRDA FD:OCA Object

To accomplish all data representation objectives, some special (DRDA-defined) usage of FD:OCA descriptors is required. Figure 5-3 shows this usage.

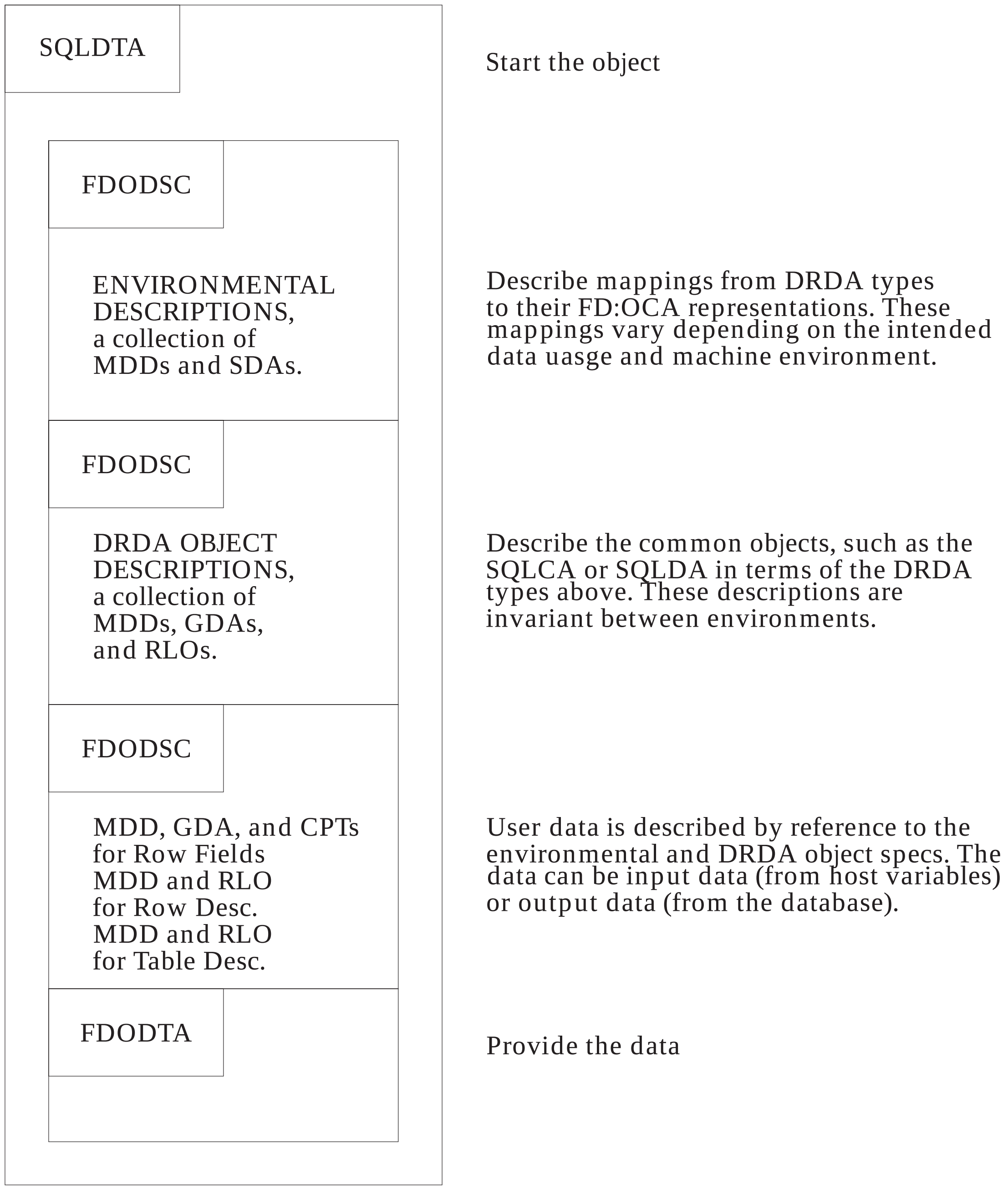

SQLDTA
Start the object
FDODSC
ENVIRONMENTAL DESCRIPTIONS, a collection of MDDs and SDAs.
Describe mappings from DRDA types to their FD:OCA representations. These mappings vary depending on the intended data uasge and machine environment.
FDODSC
DRDA OBJECT DESCRIPTIONS, a collection of MDDs, GDAs, and RLOs.
Describe the common objects, such as the SQLCA or SQLDA in terms of the DRDA types above. These descriptions are invariant between environments.
FDODSC
MDD, GDA, and CPTs for Row Fields MDD and RLO for Row Desc. MDD and RLO for Table Desc.
User data is described by reference to the environmental and DRDA object specs. The data can be input data (from host variables) or output data (from the database).
FDODTA
Provide the data

Figure 5-3 Conceptual View of a DRDA FD:OCA Object

The discussion that follows covers the concepts behind the DRDA FD:OCA objects. The FD:OCA descriptors sections are shown in DDM FDODSC carrier objects. The FD:OCA data is shown in DDM FDODTA objects. Both of these are shown as being contained in an SQLDTA carrier. When these descriptors actually flow, not all of these parts will be physically present and in many cases the carriers will be different.

The ENVIRONMENTAL DESCRIPTION section of the descriptor has a Simple Data Array (SDA) to describe how each DRDA type is represented.[35] DRDA defines an entire set of data types for each environment supported. See Section 5.6.5 on page 197 for a complete listing.

An immediately preceding Meta Data Definition (MDD) specification relates each DRDA type representation to its SDA (or GDA or RLO). DRDA defines meta data type references for each DRDA type. FD:OCA defines that MDDs apply to other triplets that follow. The following SDA, GDA, or RLO, thus, is the presentation for a particular DRDA type for this environment.

Each of the SDA, GDA, and RLO triplets is assigned a local identifier (LID) that is used as a short label for references to these triplets. Using LIDs, triplets can refer to other triplets, which in turn can refer to yet other triplets, and so on. A direct mapping from DRDA types can then be made from DRDA type to LID and back. DRDA provides named sets of descriptors that establish a firm relationship between LID and DRDA type. All types are provided in each set of environmental descriptors; the representations vary from environment to environment.

The next section of the descriptor contains DRDA OBJECT DESCRIPTIONS. Objects such as the SQLDA or SQLCA are defined in terms of the DRDA types described in the previous section. These descriptions are not sensitive to environment. Everyone uses one set of identical descriptors. However, the exact bits that flow when one implementation sends one of the described objects to another implementation vary depending on the environmental descriptors in use. These descriptors are also preceded with MDD triplets that define the DRDA semantics of the FD:OCA descriptors.

The final section contains the description of user data. In most cases, environmental and DRDA object descriptions form this description. The referenced SDAs and GDAs are assembled to reflect the order and characteristics of the user and system data that flow. In some cases, additional SDAs are required to handle data the database management system has returned. For example, if the database management system has returned data in an unusual CCSID, an SDA and an MDD (defining the DRDA semantics) are built to indicate that situation to the requester. See Section 5.6.6 on page 243 for more detail.

The organization of the FD:OCA descriptive triplets as shown in Figure 5-3 on page 140 gives the benefits of environment-independent specification of user data and commonly used information blocks. This is tailored with environment definitions that show exactly (to the bit) how each of these blocks really appears in each environment. They are different from environment to environment. However, systems that use identical type representations will exchange data with no conversion or translation.

A conceptual view of a DRDA FD:OCA object with LOB data is given below. It includes the definition and flow of externalized FD:OCA data. Figure 5-4 on page 142 shows this usage.

35. For individual fields, DRDA types map very closely to SQL data types. Exceptions occur where inconsistencies in type assignment method have occurred in SQL. For example, 4-byte and 2-byte integers are different SQL types, but 4-byte and 8-byte floating point are not. DRDA also has types for common collections of fields where SQL does not.

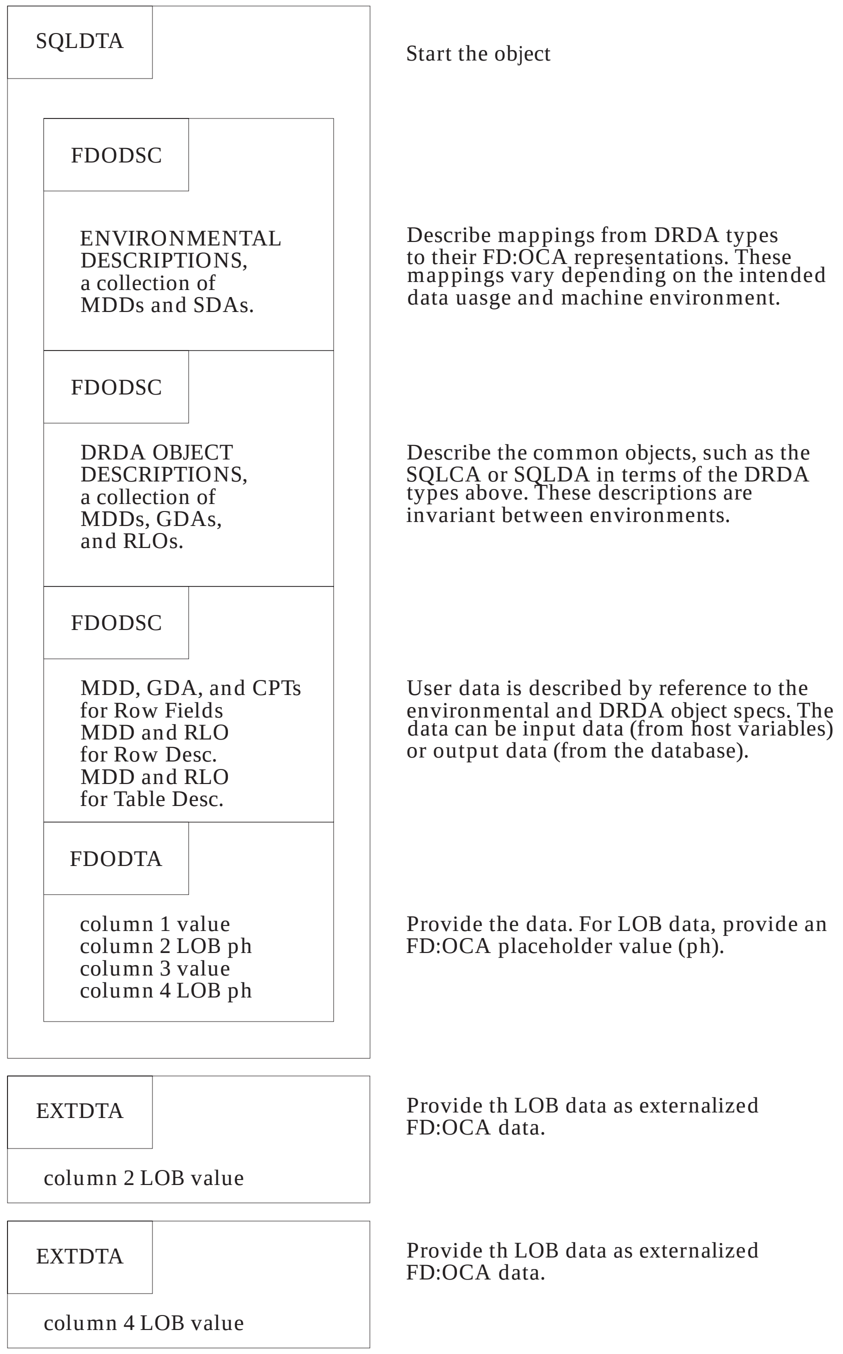

SQLDTA
Start the object
FDODSC
ENVIRONMENTAL DESCRIPTIONS, a collection of MDDs and SDAs.
Describe mappings from DRDA types to their FD:OCA representations. These mappings vary depending on the intended data uasge and machine environment.
FDODSC
DRDA OBJECT DESCRIPTIONS, a collection of MDDs, GDAs, and RLOs.
Describe the common objects, such as the SQLCA or SQLDA in terms of the DRDA types above. These descriptions are invariant between environments.
FDODSC
MDD, GDA, and CPTs for Row Fields MDD and RLO for Row Desc. MDD and RLO for Table Desc.
User data is described by reference to the environmental and DRDA object specs. The data can be input data (from host variables) or output data (from the database).
FDODTA
column 1 value
column 2 LOB ph
column 3 value
column 4 LOB ph
Provide the data. For LOB data, provide an FD:OCA placeholder value (ph).
EXTDTA
column 2 LOB value
Provide th LOB data as externalized FD:OCA data.
EXTDTA
column 4 LOB value
Provide th LOB data as externalized FD:OCA data.

Figure 5-4 Conceptual View of a DRDA FD:OCA Object with LOB Columns

This DRDA FD:OCA object contains a row with four columns, two of which are LOB columns. Note that the LOB columns are represented by FD:OCA placeholder values in FDODTA. The actual LOB data values are carried in the EXTDTA object in the order in which they appear in the FDODTA object.

5.2.3 Early and Late Descriptors

The environmental descriptors are the same for all data flowing in one direction over any conversation. At the very latest, this information could flow with the data. At the very earliest, DRDA for some set of known environments could define this information and reference it by name. The named descriptor would contain a full set of SDAs to cover all SQL data types for a particular environment. The actual FD:OCA SDAs are virtual. The products could do the proper conversions, knowing at code design time the appropriate conversions to do under each circumstance. These conversions are based strictly on the DRDA type used to represent the value without interpretation of a real SDA. That saves both implementation cost and line time.

Section 5.6.5 on page 197 defines five DRDA environments: QTDSQL370, QTDSQL400, QTDSQLX86, QTDSQLASC, and QTDSQLVAX machine representations. The TYPDEFNAM on the ACCRDB command references these environments, and the associated early descriptors never flow.

Common objects (such as SQLCAs) are the same for every product operating at the same level of DRDA. These objects can be identified early. The latest time the user needs to determine the descriptor set is at EXCSAT time. Descriptions of the common objects can be made with DRDA named sets of descriptors that relate to the DRDA level being supported. By staying within the set of DRDA defined common blocks, no runtime interpretation of FD:OCA triplets is required.

Section 5.6.4 on page 177 and the sections following the figure define these descriptors. These are agreed to at EXCSAT time by means of MGRLVL parameter for the SQL Application Manager, SQLAM X‘2407’. (See the DDM Reference for definitions of these variables.)

The descriptor of the final object is built of descriptions provided or implied at three separate times: EXCSAT, ACCRDB, and finally right before user data transmission.

Objects defined by early descriptors need only contain the data; objects defined by late descriptors must include the FD:OCA descriptor and the data. Often the DDM code point of the command or reply implies the format of the data. In other cases, the descriptor must be sent. There are three distinct cases:

1. The data format is completely implied by the DDM code point.
2. The data format varies from one instance to another of the DDM command or reply.
3. The data format varies but was defined in a preceding command or reply.

In the first case, the FD:OCA descriptor is not sent. The DDM code point relates to a DRDA-defined FD:OCA descriptor. The FD:OCA descriptors for these fixed format data are known early, and they reference the environment descriptors to set final representations. Thus, in the case of fixed command data and reply data formats, the data immediately follows the DDM code point. This case includes all commands in Table 5-1 on page 146, except for Execute SQL Statement (EXCSQLSTT), Open Query (OPNQRY), and Continue Query (CNTQRY).

The second case corresponds to DRDA transmission of database rows or database input (host variable) values. For these, the descriptor cannot be constructed until the data is presented for transmission. These descriptors are late descriptors. There are four subcases:

2a. For some commands or replies, the DDM code point enclosing the command or reply data provides a complete FD:OCA object using SQLDTA or SQLDTARD. Inside that object, the FD:OCA descriptors are sent in an FDODSC object followed by the data in an FDODTA object. This case includes the command data for Execute SQL Statement (EXCSQLSTT) and Open Query (OPNQRY). This case also applies directly to reply data when the size of the result is known in advance. This is the case for the result of Execute SQL Statement (EXCSQLSTT), which can return at most one row of result data.

If there are LOB values in the answer set, the descriptors indicate whether an FD:OCA placeholder will flow for a column, and if so, each LOB data value will flow in an EXTDTA following the SQLDTA or SQLDTARD in the order they appear in the FDODTA.

2b. For the result of Open Query (OPNQRY) and Continue Query (CNTQRY), the size of the result is not known in advance. A Query Descriptor (QRYDSC) object is built to describe the following data. The application server constructs as many Query Data (QRYDTA) objects as are necessary to contain the entire result.

If there are LOB values in the answer set when the application server processes the OPNQRY command, it assumes that LOB data values are to be returned for LOB data columns. The QRYDSC indicates that FD:OCA placeholders will flow for each LOB column. At CNTQRY time, the application requester may override the QRYDSC description by sending an OUTOVR object. In this way, the application server knows whether to send LOB data values or LOB locators for a LOB column. Because the application server does not know the desired format of the data to be returned, it does not send an QRYDTA object until the first CNTQRY is received.

2c. For an Execute SQL Statement (EXCSQLSTT) that invokes a stored procedure that returns one or more result sets, the number and the size of the result sets is not known in advance. A Query Descriptor (QRYDSC) object is built for each result set to describe the data that follows. The application server constructs as many Query Data (QRYDTA) objects for each result set as are necessary to contain the entire result.

If there are LOB values in the answer set when the stored procedure is executed, the application server assumes that LOB data values are to be returned for LOB data columns in a query result set. The QRYDSC carries FD:OCA placeholders for each LOB column. At CNTQRY time, the application requester may override the QRYDSC description by sending an OUTOVR object. In this way, the application server knows whether to send LOB data values or LOB locators for a LOB column. Because the application server does not know the desired format of the data to be returned, it does not send a QRYDTA object until the first CNTQRY is received.

2d. If there are LOB data columns in the output of a command, then the Output Override Descriptor Object (OUTOVR) may be sent with the command to specify the format of the LOB columns. The command may either be a Continue Query (CNTQRY) requesting additional rows in a query result set, or an Execute SQL Statement (EXCSQLSTT) where the statement is not a stored procedure invocation.

The third case corresponds to the continuation of an interrupted set of rows in response to a query or the execution of a stored procedure, such as the response to Continue Query (CNTQRY). In this case, it is not necessary to describe the format of the rows being sent again because the format is the same as the format of the rows of the query that were already sent using the second form of DDM/FD:OCA data description and transmission. Therefore, the receiver of a query result must retain the data description sent in response to the Open Query and associate that description with the opened query.

5.3 Relationship of DRDA and DDM Objects and Commands

This section describes the relationship between DRDA and DDM objects and commands.

5.3.1 DRDA Command to Descriptor Relationship

Data objects defined by DDM for DRDA can contain command data or reply data described by either early or late FD:OCA descriptors. For the SQLCARD, SQLDARD, SQLPARD, SQLRSLRD, SQLCINRD, SQLSTT, SQLSTTVRB, and SQLOBJNAM, the description of the data is completed by the time ACCRDB completes. In these cases, the early descriptors are sufficient to define the data that is flowing. The SQLDTA and SQLDTARD contain descriptors and data defined by those late descriptors. The QRYDTA contains data defined by the QRYDSC late descriptor.

One or more FDODSCs or QRYDSCs are required to describe the data, and one or more FDODTAs or QRYDTAs are required to contain the data.[36] An FDODSC and an FDODTA are contained within SQLDTA and SQLDTARD. When a QRYDSC or a QRYDTA is used, one of each is all that is logically required. However, with the small block sizes DRDA allows, several DDM objects can be required to contain the whole description or data. Also, transmission of the data can begin before the entire result has been fetched from the database, so the result will be sent in pieces. TYPDEFNAM and/or TYPDEFOVR can precede command data or reply data objects as environmental overrides.

Table 5-1 on page 146 shows data associated with each DRDA command described in Section 4.3.1.11 on page 43. All descriptors named here are described later in this chapter.

Table 5-1 on page 146 consists of five columns. The first column names the command being described. The second states whether command data (from application requester to application server) or reply data (from application server to application requester) is described; therefore, there are two rows for each command. The third column names the DDM carrier object, which is a DDM code point defined in the DDM Reference. It will contain information described by the DRDA descriptor named in the fourth column. In most cases, the third and fourth columns are the same. In cases where several different DDM commands are required to carry the DRDA object, these names will not match. This is most often the case when a command requires both descriptor and data objects to flow on the link, and the DRDA object is split over command boundaries. Note that CNTQRY has only the QRYDTA carrier because the descriptor has been completely carried in the preceding OPNQRY or EXCSQLSTT command. The DRDA query result will flow in response to an OPNQRY and zero or more CNTQRY commands. One or more DRDA stored procedure result sets will flow in response to an EXCSQLSTT and zero or more

36. Although QRYDSC and SQLDARD both provide a description of the data, they are for different purposes and should not be confused with each other. QRYDSC is part of the result of an OPNQRY or EXCSQLSTT to be used internally to describe the rows being returned. SQLDARD is the result of a PRPSQLSTT or DSCSQLSTT and is mapped to an external SQLDA expected by the application program. Only QRYDSC and not SQLDARD must be used as the basis of data representation for the data returned by OPNQRY, EXCSQLSTT, and CNTQRY.

CNTQRY commands. The fifth column is a description of the data content of the object.

Table 5-1 Data Objects, Descriptors, and Contents for DRDA Commands

DRDA Command	Command or Reply Data	DDM Object Name	DRDA Descriptor Name	Data Content Description
ACCRDB	Command	None.	None.	None.
ACCRDB	Reply	None. or SQLCARD	None. or SQLCARD	None. Return Code / Status
BGNBND	Command	None.	None.	None.
BGNBND	Reply	SQLCARD	SQLCARD	Return Code / Status
BNDSQLSTT	Command	SQLSTT and SQLSTTBRV	SQLSTT and SQLSTTVRB	Modified SQL Statement Description of Variables that appeared in the Statement
BNDSQLSTT	Reply	SQLCARD	SQLCARD	Return Code / Status
ENDBND	Command	None.	None.	None.
ENDBND	Reply	SQLCARD	SQLCARD	Return Code / Status
DRPPKG	Command	None.	None.	None.
DRPPKG	Reply	SQLCARD	SQLCARD	Return Code / Status
REBIND	Command	None.	None.	None.
REBIND	Reply	SQLCARD	SQLCARD	Return Code / Status
PRPSQLSTT	Command	SQLSTT	SQLSTT	SQL Statement (No Variable References)
PRPSQLSTT	Reply	SQLCARD or SQLDARD	SQLCARD or SQLDARD	Return Code / Status Result Row Description including Labels
EXCSQLSTT (Notes 1, 2)	Command	SQLDTA or OUTOVR and SQLDTA or SQLDTA and EXTDTA	SQLDTA or SQLDTAMRW SQLDTA SQLDTA or SQLDTAMRW SQLDTA or SQLDTAMRW	Data Descriptors and Values Data Descriptors and Values

DRDA Command	Command or Reply Data	DDM Object Name	DRDA Descriptor Name	Data Content Description
		or		
		OUTOVR	SQLDTA	
		and		
		SQLDTA	SQLDTA	
			or	
			SQLDTAMRW	
		and		
		EXTDTA		
EXCSQLSTT	Reply	SQLCARD	SQLCARD	Return Code / Status
(Notes 2,3,4,5)		and	and	
		SQLRSLRD	SQLRSLRD	Information about Result Sets
		and	and	
		SQLCINRD	SQLCINRD	Column Information for a Result Set
		and	and	
		QRYDSC	SQLDTARD	Reply Data Descriptor and Values
		and		
		QRYDTA		
		or	or	
		SQLDTARD	SQLDTARD	Return Code / Status and
		and	and	Output Parameter Values
		SQLRSLRD	SQLRSLRD	Information about Result Sets
		and	and	
		SQLCINRD	SQLCINRD	Column Information for a Result Set
		and	and	
		QRYDSC	SQLDTARD	Reply Data Descriptor and Values
		and		
		QRYDTA		
		or	or	
		SQLDTARD	SQLDTARD	Return Code / Status and
		and	and	Output Parameter Values
		SQLRSLRD	SQLRSLRD	Information about Result Sets
		and	and	
		SQLCINRD	SQLCINRD	Column Information for a Result Set
		and	and	
		QRYDSC	SQLDTARD	Reply Data Descriptor and Values
		and		
		EXTDTA		Externalized FD:OCA Data
		or	or	
		SQLDTARD	SQLDTARD	Reply Data Descriptor and Values
		or	or	
		SQLDTARD	SQLDTARD	Reply Data Descriptor and Values
		and		
		EXTDTA		Externalized FD:OCA Data
EXCSQLIMM	Command	SQLSTT	SQLSTT	SQL Statement (No Variable References)

DRDA Command	Command or Reply Data	DDM Object Name	DRDA Descriptor Name	Data Content Description
EXCSQLIMM	Reply	SQLCARD	SQLCARD	Return Code/Status
DSCSQLSTT	Command	None.	None.	None.
DSCSQLSTT	Reply	SQLCARD or SQLDARD	SQLCARD or SQLDARD	Return Code/Status Result Row or Input Parameter Description Including Labels
DSCRDBTBL	Command	SQLOBJNAM	SQLOBJNAM	SQL Table Name
DSCRDBTBL	Reply	SQLCARD or SQLDARD	SQLCARD or SQLDARD	Return Code/Status Table Description
OPNQRY (Note 2)	Command	SQLDTA or SQLDTA and EXTDTA	SQLDTA or SQLDTA EXTDTA	Parameter Descriptor and Values
OPNQRY (Note 6)	Reply	SQLCARD or QRYDSC and QRYDTA	SQLCARD or SQLDTARD	Return Code/Status Reply Data Descriptors and Values
CNTQRY (Note 2)	Command	OUTOVR	SQLDTA	Output Override Descriptor
CNTQRY (Note 2)	Reply	SQLCARD or QRYDTA or QRYDTA and EXTDTA	SQLCARD or SQLDTARD or SQLDTARD	Return Code/Status Reply Data Descriptor and Values Reply Data Descriptor and Values
CLSQRY	Command	None.	None.	None.
CLSQRY	Reply	SQLCARD	SQLCARD	Return Code/Status
RDBCMM	Command	None.	None.	None.
RDBCMM	Reply	SQLCARD	SQLCARD	Return Code/Status
RDBRLLBCK RDBRLLBCK	Command Reply	None. SQLCARD	None. SQLCARD	None. Return Code/Status
EXCSQLSET	Command	None.	None.	None.
EXCSQLSET	Reply	SQLCARD	SQLCARD	Return Code/Status

Notes:

1. SQLDTAMRW is not supported in DRDA Level 1.
2. EXTDTA and OUTOVR are not supported in DRDA Levels 1, 2 or 3.
3. SQLRSLRD and SQLCINRD are not supported in DRDA Levels 1 or 2.

4. If any LOB data is in the CALL parms, then the associated EXTDTAs must be the last in the reply chain, so must flow after the result set objects.
5. If any LOB data is in a result set, then no QRYDTA is returned until the first CNTQRY for the result set.
6. If any LOB data is in the query, then no QRYDTA is returned until the first CNTQRY.

SQLDTA, SQLDTAMRW (SQLDTAMRW is not supported in DRDA Remote Unit of Work), and SQLDTARD are the only late descriptors in Table 5-1 on page 146. SQLDTA, SQLDTAMRW, and SQLDTARD have a dependency on the late descriptor, SQLDTAGRP. SQLDTARD also has a dependency on SQLCADTA, which is another late descriptor. These are the only ones that must be transmitted by FDODSC, or QRYDSC. Early descriptors and flows describe all other command and reply data as stand-alone data in the appropriate DDM object.

5.3.2 Descriptor Classes

FD:OCA provides a powerful and flexible mechanism to model data or collections of data. To describe DRDA objects, the FD:OCA constructs Simple Data Array (SDA), Group Data Array (GDA), and Row Layout (RLO) triplets are used. Each SDA, GDA, and RLO is assigned, through the Meta Data Definition triplet (MDD), a unique DRDA type. In the case of SDAs, the DRDA type is always a data type that DRDA supports. Each group is assigned a DRDA type and describes an ordered collection of other groups or Simple Data Arrays (possibly including length overrides). A row is assigned a DRDA type and describes an ordered set of elements, each of which is selected from one or more groups. An array is assigned a DRDA type and describes a finite number of rows.

DRDA has four classes of descriptors that participate in defining user data. These classes are described below:

1. **Environmental descriptors** show how each SQL and DRDA data type are represented on the link. These descriptors are built from FD:OCA Meta Data Definition (MDD) triplets and Simple Data Array (SDA) triplets.

   These descriptors set maximum limits for lengths and indicate how floating point numbers should be represented. The SDAs also represent integer data, such as byte reversed.

2. **Group descriptors** instantiate one or more fields into a collection or group. Groups can be nullable as a whole, independent of the nullability of the individual component fields. These descriptors are built from FD:OCA MDDs and GDA triplets (and Continue Preceding Triplet (CPT) if needed).

   These descriptors provide overrides for lengths or precision and scale of previously specified environmental descriptors. For example, the general fixed decimal specification allows up to 31 digits. Those 31 digits can be to the right of the decimal point. The group descriptor specifies the actual values for some particular instance of a fixed decimal number; for example, 5 digits with 2 to the right of the decimal point. A group of fields, so defined, acquires a local identifier (LID) and can be subsequently referenced in later descriptors by name.

   DRDA requires a length override (non-zero value) for each referenced environmental descriptor for output data (data originating at the application server) to optimize the amount of storage allocated. The default SDA length for character data is 32767, which allows an override up to that value. If no override is given, storage for the default 32767 might have to be allocated, even though space for one character can be all that is needed.

To form nullable sets of fields, a group descriptor is used. A nullable group provides one indicator byte that indicates the presence or absence of the whole group. The triplets that such a descriptor references can be Environmental Descriptor SDAs (in which case the overrides described above are applied as well) or other Group Descriptor GDAs (in which case no overrides occur). The referenced GDAs can be either nullable or non-nullable.

3. **Row descriptors** instantiate a row of fields. Each row, like a row in a relational table, has the same number of fields represented. Some fields can be null; some groups of fields can be null (with just one null indicator); however, all fields are accounted for. These descriptors are built from FD:OCA MDDs and RLO triplets.

   The rows are constructed from previously specified groups. Where the group provided specific length information about each field, the row strings the fields out into a one dimensional vector. The groups that become part of the row can be a mixture of objects. For example, the user data values that are returned as the result of a query are carried in a row containing an SQLCA as well as the user data.

4. **Array descriptors** define open ended data structures. SQLDAs and user data are organized as open ended repetitions of column descriptions and table data. These descriptions make rows into tables, the size of which is determined by the amount of data that follows. These descriptors are built from FD:OCA MDDs and RLO triplets.

   References to row descriptors build these descriptors. The descriptors take one dimensional vectors or rows and produce two dimensional tables. In the previous example, the entire query result would be an array. There would be as many rows in the array as there were rows in the answer set. Each of these rows would be the special SQLCA/user data hybrid described above. (Nullability of the SQLCA group allows it to be transmitted as a single byte in the normal case.)

The relationship between these DRDA classes is such that FD:OCA triplets of any class can reference descriptors of the next lower numbered class only. This consistently maintains the dimensionality of each class. The only exceptions to this next-lower rule are GDAs that build Group Descriptors can reference both Environmental Descriptors and other Group Descriptors; the result is still a group. Each of these classes corresponds to a meta data type used in MDD descriptors for relational databases.

In addition, to comply with FD:OCA reference rules, all FD:OCA triplets referenced by any triplet must precede that triplet. Therefore, all environmental triplets must precede the group descriptor that references them. Similarly, all group triplets must precede the row triplets, and these must precede the array describing triplets. See examples in Section 5.8.2 on page 259.

The early descriptors never actually flow on the link. The Environmental Descriptors are determined by ACCRDB processing by means of TYPDEFNAM. The Group, Row, and Array descriptors are agreed to during EXCSAT processing by means of the MGRLVL parameter for the relational database manager.

The late descriptors physically flow on the link. These FD:OCA descriptors are always contained within a DDM FDODSC, or QRYDSC. (See the blocking discussion in Chapter 7 on page 281.)

The data inside an FDODSC, or QRYDSC is always presented in high to low order byte ordering. Other machines that use byte reversed numbers must translate the data because the numbers are not byte reversed. There are no alphabetics, so CCSID is of no concern. There is also no floating point data, so that is of no concern. The application requester and application server must send the data exactly as shown in these examples. See the DDM Reference for more details and diagrams.

5.4 DRDA Descriptor Definitions

Section 5.3.2 on page 149 described the logical dependency and physical ordering of the descriptor triplets. This order is the proper sequence. First, define the basic descriptor building blocks, assemble them into larger descriptor components, and finally assemble the descriptor needed.

However, for DRDA, there are a large number of basic descriptor building blocks (such as the data type information), which can obscure understanding of how descriptors are built. To avoid this confusion, descriptor assembly will be explained using a top-down approach. Beginning with the end product, which is the final descriptor, assembly will be broken down into its component parts. Late arrays are discussed first, followed by late rows and late groups and then early arrays, rows, and groups are presented. The environmental descriptors, early and late, are discussed last. Until implementation time, the fine details of data types and machine representation are not needed.

5.5 Late Descriptors

One class of DRDA objects' descriptions are not known at connection time, but can only be known at SQL statement execution time. These include descriptions of input host variables passed by the application in support of OPNQRY and EXCSQLSTT, and descriptions of the answer set returned in response to an OPNQRY or an EXCSQLSTT for an SQL static SELECT or for an SQL CALL that invokes a stored procedure. In these cases, the number of host variables or columns and their SQL data types and lengths are only known when the application executes the statement. The description of the data is assembled dynamically and sent to the application requester/application server along with (but preceding) the data. These are called late descriptors.

The DRDA types are fixed at ACCRDB/ACCRDBRM processing. The SQL types of all input host variables and constituent columns of an answer set must be mappable to one of these DRDA types.

5.5.1 Late Array Descriptors

The following figures describe DRDA defined Late Array Descriptors. These array descriptors are built on the one-dimensional row descriptors defined in Section 5.5.2 on page 155. These array descriptors add one dimension to the structure of the objects (defined by rows). These objects are all constructed with the FD:OCA Row Layout (RLO) triplet. Each descriptor consists of a single RLO.

The format of these descriptors is described in Figure 5-5 on page 153 and Figure 5-6 on page 155. Each DRDA type consists of a Meta Data Definition (MDD) that states the DRDA semantics of the descriptor followed by one RLO triplet that refers to the other RLO triplets.

The result is the definition of a two-dimensional object. This object is a logical array of information. It can begin with a fixed number of occurrences of zero or more formats of lower level rows. It can end with an indefinite number of occurrences of a single row format.

5.5.1.1 SQLCA with Data Array Description

Meta Data and Data Descriptor for SQLDTARD

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X’78’	0	Rel.DB X’05’	ArrayUnit X’04’	Next Byte X’01’	SQLDTARD X’F0’

Length 0	Type 1	Identity 2
6	RLO X’71’	X’F0’

Descriptor Reference:

SQLCADTA

Row LID	Elem.Takn	RepFactor
X’E0’	0 (all)	0 (all)
1-byte	1-byte	1-byte

Descriptor in Hex: 07780005 0401F0
0671F0E0 0000

Figure 5-5 SQLDTARD Array Descriptor (SQLCAs and Data as Reply Data)

These are the descriptions for the Late Array Descriptor parameters shown in Figure 5-5.

Upper Section of Figure

The Meta Data Definition applies to the descriptor that follows it. For DRDA, it specifies exactly what the descriptor is used for. The same FD:OCA triplet can be used for several purposes.

byte 0 The length of the FD:OCA MDD triplet is always 7 for DRDA.

byte 1 Meta Data Definition type indicator is always X‘78’ for MDDs.

byte 2 The Local Identifier for the MDD is always 0.

byte 3 Application Class for the MDD. The relational database is class X‘05’ for DRDA. This byte is always X‘05’.

byte 4 Meta data type for the MDD is defined within the application class. DRDA has defined four data types as described in Section 5.7 on page 245.

byte 5 Meta data reference type for the MDD. The DRDA style of reference is the next byte contains the desired information. This is always X‘01’ for DRDA MDDs.

byte 6 Meta data reference value for this MDD. DRDA uses this value as the DRDA Type indicator. Table 5-12 on page 245 describes the acceptable values.

Middle Section of Figure

In the middle section is an RLO triplet that refers to other RLO triplets. The RLO has a header section (the small box with three parts) that contains the length, FD:OCA type, and LID for the RLO.

The section below that (another box with three parts) can be short or long. It contains one or more occurrences of the RLO's repeating group, one occurrence for each row type to be included in the final array.[37]

For each occurrence of a reference to a row descriptor RLO (a line in the box), there is a label associated with the field, a pointer to the appropriate RLO, a number of elements taken parameter (always 0 indicating that all of the elements of the row should be taken), and a repetition factor. The repetition factor can be 0 for the last RLO reference. This factor indicates that the final row appears an indefinite number of times. The actual number is not known until the data is available.

Note: The page number of the definition of the referenced RLO appears to the right of the box.

Descriptor in hex

The Descriptor in Hex section shows a fully constructed descriptor as an example.

37. This SQLDTARD descriptor has only one type of row. However, this description also applies to Section 5.6.2 on page 162, which does, in fact, have multiple row types.

5.5.1.2 Data Array Description for Multi-Row Inserts

The SQLDTAMRW structure is not supported in DRDA Level 1.

Meta Data and Data Descriptor for SQLDTAMRW

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X’78’	0	Rel.DB X’05’	ArrayUnit X’04’	Next Byte X’01’	SQLDTAMRW X’F4’

Length 0	Type 1	Identity 2
6	RLO X’71’	X’F4’

Descriptor Reference:

SQLDTA

Row LID	Elem.Takn	RepFactor
X’E4’	0 (all)	0 (all)
1-byte	1-byte	1-byte

Descriptor in Hex: 07780005 0401F4
0671F4E4 0000

Figure 5-6 SQLDTAMRW Array Descriptor (Multi-Row Insert Data)

5.5.2 Late Row Descriptors

This section describes DRDA-defined Late Row Descriptors. These objects are all constructed with the FD:OCA Row Layout (RLO) triplet and result in a one dimensional structure.

The format of these descriptors is only slightly different from the array descriptors. The top and bottom are just like the late array descriptors (see Figure 5-5 on page 153). However, in the middle there is one RLO triplet that refers to one or more group descriptors (GDAs) described in Section 5.5.3 on page 157 and in Section 5.6.4 on page 177.

For each occurrence of a reference to a group descriptor GDA (a line in the box), there is a label associated with the field, a pointer to the appropriate GDA, a parameter containing a count of elements taken (always 0 indicating that all of the elements of the group should be taken), and a repetition factor (always 1 indicating that exactly one occurrence of the group should be taken).

The result is the definition of a one dimensional object, a row or vector, or a control block without any repeating groups.

5.5.2.1 Row Description for One Data Row

Meta Data and Data Descriptor for SQLDTA

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Row Unit X'03'	Next Byte X'01'	SQLDTA X'E4'

Length 0	Type 1	Identity 2
6	RLO X'71'	X'E4'

Descriptor Reference:

SQLDTAGRP

Row LID	Elem.Takn	RepFactor
X'D0'	0 (all)	1
1-byte	1-byte	1-byte

Descriptor in Hex: 07780005 0301E4
0671E4D0 0001

Figure 5-7 SQLDTA Row Descriptor

The SQLDTA describes all Input, Update, or Parameter fields for a single SQL statement or the fields for one row of result data.

This descriptor carries type information (by SDA references from the SQLDTAGRP descriptor) and length, precision, and scale information (in the SQLDTAGRP descriptor) and is packaged as a single block (row).

5.5.2.2 *Row Description for One Row with SQLCA and Data*

Meta Data and Data Descriptor for SQLCADTA

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Row Unit X'03'	Next Byte X'01'	SQLCADTA X'E0'

Length 0	Type 1	Identity 2
9	RLO X'71'	X'E0'

Descriptor Reference:	Group LID	Elem.Takn	RepFactor
SQLCAGRP	X'54'	0 (all)	1
SQLDTAGRP	X'D0'	0 (all)	1
	1-byte	1-byte	1-byte

Descriptor in Hex: 07780005 0301E0
0971E054 0001D000 01

Figure 5-8 SQLCADTA Row Descriptor (One Row with SQLCA and Associated Data)

5.5.3 Late Group Descriptors

The group descriptors that follow collect field definitions together, specify length attributes, provide ordering of the fields, provide nullability of the collection (with one null indicator), and provide a local identifier (LID) for the group. The groups are all constructed with the FD:OCA Group Data Array (GDA) triplet. For large groups (more than 84 fields), Continue Preceding Triplet (CPT) is used repeatedly, as necessary, to contain enough GDA repeating groups.

The format of these descriptors is only slightly different from the row descriptors. The top and bottom are the same as Figure 5-5 on page 153. However, in the middle is one GDA descriptor (with 0 or more CPT triplets) where there was one RLO triplet. The GDA has a header section (the small box with 3 parts) containing the length, FD:OCA type, and LID for the GDA. The box below that (a box with 2 parts) can be short or long. It contains one or more occurrences of the GDA repeating group, one occurrence for each field to be included in the group.

For each occurrence of a reference to an environmental SDA (a line in the box), there is a label associated with the field, a pointer to the appropriate SDA, and the overriding length parameter.

The overriding length parameter is a 2-byte field.

- For all FD:OCA data types, except FD:OCA Generalized Strings, the last 15 bits of the overriding length parameter is a signed 2-byte integer indicating the length of the data described, according to the FD:OCA type of the data.

  The first bit is '0'b.

- For FD:OCA Generalized String data types, the overriding length parameter is a signed 2-byte integer indicating the length of the length portion of the data.

The first bit is the FD:OCA placeholder indicator flag.

— If the FD:OCA placeholder indicator flag is set on ('1'b), the DRDA object carrying the data described by the SQLDTAGRP contains only the length portion of the generalized string. This data acts as FD:OCA placeholder for the value portion of the FD:OCA Generalized String which is itself externalized to another DRDA object called the EXTDTA.

  The overriding length field gives the number of bytes in the FD:OCA placeholder.

— The FD:OCA placeholder indicator flag may not be set off ('0'b) in DRDA Level 4 for this type of data. Thus, all such data must be externalized and must flow in an EXTDTA.

  DRDA Level 4 specifies a length override value of 8, indicating that allowable placeholder sizes are 2, 4, 6, or 8. The placeholder size is the minimum needed to hold the length value for the largest possible data value. Thus, an FD:OCA Generalized String item whose maximum length is less than 32767 has a 2-byte placeholder, while one whose maximum length is less than 2147483647 has a 4-byte placeholder, and so on.

  An EXTDTA object must flow for each FD:OCA Generalized String described, except if the data is a nullable data item and the data is null or the data has a zero length (a placeholder value of zero).

All of the numbers in the boxes are in decimal unless otherwise noted. See Section 5.6.6 on page 243 and Section 5.6.5 on page 197 for a discussion of environmental descriptors. The length overrides are required (must not be zero) when referring to SDAs for output data (data originating at the application server).

5.5.3.1 SQL Data Value Group Description

Meta Data and Data Descriptor for SQLDTAGRP

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	GroupUnit X'02'	Next Byte X'01'	SQLDTAGRP X'D0'

Length 0	Type 1	Identity 2
3N + 3 (N < 85)	N-GDA X'76'	X'D0'

Column Name or User Data Label:

	Env. LID	Length Override
Column 001	as req'd	as req'd
. . .	as req'd	as req'd
Column x	as req'd	as req'd

Note: Continue Preceding Triplet Needed to cover more than 84 columns. CPT is optional otherwise.

Length 0	Type 1	Identity 2
3N + 3 (N < 85)	CPT X'7F'	0

	Env. LID	Length Override
Column x+1	as req'd	as req'd
. . .	as req'd	as req'd
Column last	as req'd	as req'd
	1-byte	2-bytes

```
Descriptor in Hex:  07780005   0201D0
                    ..76D0..   ........   ........   ........
                    ..7F00..   ........   ........
```

Figure 5-9 SQLDTAGRP Group Descriptor (Field Specs for One Row of SQL Data)

The dots for the descriptor in hex indicate that the values are not known until runtime.

5.5.3.2 Overriding Output Formats

For most output columns, the target system returns data for the column in a format determined by the target. This format will be designated as the default format for the output column and is dependent on the data type of the column. The default format is described in an FDODSC or QRYDSC object returned by the target to the source system. The source system is responsible for performing numeric conversions and character translations from the default format to the actual format desired by the application.

This protocol is expanded for LOB data since applications may request that a LOB data column be returned either as value bytes or as a locator value. Since a locator value is generated by the target to represent the LOB column, the source system can only return a locator value to the application if the target system generated it and returned it in that format.

The default format for a LOB output column is as data value bytes. If the application wants to receive a locator value for the column, the source system sends a descriptor object, called the OUTOVR command data object, with the command that returns LOB data columns as output.

The OUTOVR object contains descriptors that override the format of output data columns. It consists of an SQLDTARD descriptor, including an SQLDTAGRP which overrides each output column as follows:

- Each column whose format is not to be overridden is represented in the SQLDTAGRP by a triplet consisting of a LID value of zero and a length override value of zero. Each such triplet is known as a *default triplet*.
- Each column whose format is to be overridden is represented by a valid triplet. Each such triplet is known as an *override triplet*.
- Only LOB locator LIDs may be specified in an override triplet.
- Each column in the output must be represented in the SQLDTAGRP by either a default triplet or an override triplet.
- The *i*th triplet overrides the *i*th output column.

  When an OUTOVR object is received with a command that returns output, then:

  — If the *i*th triplet is a default triplet, then the *i*th column in the output is returned in the default format.

  — If the *i*th triplet is an override triplet, then the *i*th column in the output is returned in the override format, if it is valid.

The following QRYDSC describes an answer set with three columns, an NFCS column, an NOCS column, and an NRI column.

Table 5-2 QRYDSC with Default Formats

Reference	Hex Representation	Description
QRYDSC	001F241A	DDM codepoint
SQLDTAGRP	0C76D0	Start nullable group descriptor
SQLDTAGRP	310014CB 00021F00 28	Continue—one CHAR, one CLOB, and one ROWID column
SQLCADTA	0971E0	Start row descriptor
SQLCADTA	540001	Continue—one group X'54'
SQLCADTA	D00001	Continue—one group X'D0'
SQLDTARD	0671F0	Start array descriptor
SQLDTARD	E00000	Continue—all row X'E0'

To override the NOCS column (X'CB' LID) in the above QRYDSC with an NOCL format (X'1B' LID), the following OUTOVR object is sent with the CNTQRY command:

Table 5-3 OUTOVR with One Override Triplet

Reference	Hex Representation	Description
OUTOVR	001F2415	DDM codepoint
SQLDTAGRP	0C76D0	Start nullable group descriptor
SQLDTAGRP	0000001B 00040000 00	Continue—one default triplet, one override triplet for NOCL, one default triplet
SQLCADTA	0971E0	Start row descriptor
SQLCADTA	540001	Continue—one group X'54'
SQLCADTA	D00001	Continue—one group X'D0'
SQLDTARD	0671F0	Start array descriptor
SQLDTARD	E00000	Continue—all row X'E0'

5.6 Early Descriptors

During application requester to application server connection processing, a subset of the DRDA object descriptions is fixed and cannot be changed for the duration of that connection. These descriptors are called the DRDA early descriptors and consist of Simple Data Arrays (SDAs), Group Data Arrays (GDAs), rows and arrays (RLOs). The Simple Data Arrays describe each data type supported by DRDA and are called the Early Environmental Descriptors. Additionally, the early descriptors include a small number of groups, rows, and arrays (GDAs and RLOs). Once the connection process has been established, the application requester/application server need only send the DRDA object to the application server/application requester; the descriptor will not be sent.

An application requester or application server commits support to the early descriptors at two distinct points during connection processing:

1. The Early DRDA Group, Row, and Array Descriptors are established during EXCSAT/EXCSATRD processing by the manager level (MGRLVL) of the SQLAM.
2. The Early Environmental Descriptors (SDAs) are established during ACCRDB/ACCRDBRM processing by TYPDEFNAM and TYPDEFOVR.

These descriptors represent the supported DRDA types and are fixed and identical across all five environments: QTDSQL370, QTDSQLX86, QTDSQL400, QTDSQLASC, and QTDSQLVAX. Accepting one of these environments commits the application requester or application server to support all DRDA types in a specific machine representation. While the DRDA types cannot change, data type representations can be changed at various points in the processing of commands and replies.

5.6.1 Initial DRDA Type Representation

The DRDA type representations are initially established from TYPDEFNAM and TYPDEFOVR on ACCRDB/ACCRDBRM. These are required parameters, and there are no defaults. The representation of numeric DRDA types is defined by the TYPDEFNAM parameter, while the representation of character data (Single Byte, Mixed, Graphic) is defined by the CCSIDs specified by the TYPDEFOVR parameter. Once the DRDA data type representations have been resolved (TYPDEFNAM and TYPDEFOVR), the Early Environmental Descriptors are complete. Command and reply data can then be assembled or parsed subject to those representations.

5.6.2 Early Array Descriptors

Figure 5-12 on page 165 to Figure 5-21 on page 174 describe DRDA-defined Early Array Descriptors. These are similar to the late array descriptors in that they make two dimensional structures from one dimensional ones.

The arrays described contain database management system and application parameter information. They describe only structures that the database management system knows of well in advance. Because of this early understanding, these descriptors do not flow on the link. Rather they are the foundation for DRDA Command and Reply data objects.

5.6.2.1 SQLRSLRD Array Description

Meta Data and Data Descriptor for SQLRSLRD

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	ArrayUnit X'04'	Next Byte X'01'	SQLRSLRD X'7F'

Length 0	Type 1	Identity 2
9	RLO X'71'	X'7F'

Descriptor Reference:	Row LID	Elem.Takn	RepFactor
SQLNUMROW	X'68'	0 (all)	1
SQLRSROW	X'6F'	0 (all)	0 (all)
	1-byte	1-byte	1-byte

Descriptor in Hex:

```
07780005   04017F
09717F68   000116F00  00
```

Figure 5-10 SQLRSLRD Array Descriptor

The SQLRSLRD Array Descriptor figure describes information about the result sets contained within the reply data of EXCSQLSTT.

5.6.2.2 SQLCINRD Array Description

Meta Data and Data Descriptor for SQLCINRD

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X’78’	0	Rel.DB X’05’	ArrayUnit X’04’	Next Byte X’01’	SQLCINRD X’7B’

Length 0	Type 1	Identity 2
9	RLO X’71’	X’7B’

Descriptor Reference:	Row LID	Elem.Takn	RepFactor
SQLNUMROW	X’68’	0 (all)	1
SQLCIROW	X’6B’	0 (all)	0 (all)
	1-byte	1-byte	1-byte

Descriptor in Hex:

```
07780005   04017B
09717B68   00016B00  00
```

Figure 5-11 SQLCINRD Array Descriptor

The SQLCINRD Array Descriptor figure describes column name information for result sets contained within the reply data of EXCSQLSTT.

5.6.2.3 SQLSTTVRB Array Description

Meta Data and Data Descriptor for SQLSTTVRB

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	ArrayUnit X'04'	Next Byte X'01'	SQLSTTVRB X'7E'

Length 0	Type 1	Identity 2
9	RLO X'71'	X'7E'

Descriptor Reference:

	Row LID	Elem.Takn	RepFactor
SQLNUMROW	X'68'	0 (all)	1
SQLVRBROW	X'6E'	0 (all)	0 (all)
	1-byte	1-byte	1-byte

```
Descriptor in Hex:   07780005   04017E
                     09717E68   00016E00  00
```

Figure 5-12 SQLSTTVRB Array Descriptor

The SQLSTTVRB Array Descriptor figure describes the variables appearing in an SQL statement.

5.6.2.4 SQLCA with SQLPA Array Description

Meta Data and Data Descriptor for SQLPARD

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	ArrayUnit X'04'	Next Byte X'01'	SQLPARD X'78'

Length 0	Type 1	Identity 2
12	RLO X'71'	X'78'

Descriptor Reference:	Row LID	Elem.Takn	RepFactor
SQLCARD	X'64'	0 (all)	1
SQLNUMROW	X'68'	0 (all)	1
SQLPAROW	X'66'	0 (all)	0 (all)
	1-byte	1-byte	1-byte

Descriptor in Hex: 07780005 040178
0C717864 00016800 01660000

Figure 5-13 SQLPARD Array Descriptor (SQLCA Followed by an SQLPA as Reply Data)

5.6.2.5 SQLCA with SQLDA Array Description

Meta Data and Data Descriptor for SQLDARD

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	ArrayUnit X'04'	Next Byte X'01'	SQLDARD X'74'

Length 0	Type 1	Identity 2
12	RLO X'71'	X'74'

Descriptor Reference:	Row LID	Elem.Takn	RepFactor
SQLCARD	X'64'	0 (all)	1
SQLNUMROW	X'68'	0 (all)	1
SQLDAROW	X'60'	0 (all)	0 (all)
	1-byte	1-byte	1-byte

```
Descriptor in Hex:   07780005   040174
                     0C717464   00016800  01600000
```

Figure 5-14 SQLDARD Array Descriptor (SQLCA Followed by an SQLDA as Reply Data)

5.6.3 Early Row Descriptors

The next figures describe DRDA-defined Early Row Descriptors. These define one dimensional rows or vectors of information that the database management system understands.

5.6.3.1 SQL Result Set Description

Meta Data and Data Descriptor for SQLRSROW

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Row Unit X'03'	Next Byte X'01'	SQLRSROW X'6F'

Length 0	Type 1	Identity 2
6	RLO X'71'	X'6F'

Descriptor Reference:

SQLRSGRP

Group LID	Elem.Takn	RepFactor
X'5F'	0 (all)	1
1-byte	1-byte	1-byte

Descriptor in Hex: 07780005 03016F
06716F5F 0001

Figure 5-15 SQLRSROW Row Descriptor (Information for One Result Set)

5.6.3.2 SQL Result Set Column Information Description

Meta Data and Data Descriptor for SQLCIROW

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel;DB X'05'	Row Unit X'03'	Next Byte X'01'	SQLCIROW X'6B'

Length 0	Type 1	Identity 2
6	RLO X'71'	X'6B'

Descriptor Reference:

SQLCIGRP

Group LID	Elem.Takn	RepFactor
X'5B'	0 (all)	1
1-byte	1-byte	1-byte

Descriptor in Hex: 07780005 03016B
06716B5B 0001

Figure 5-16 SQLCIROW Row Descriptor (Information for One Column)

5.6.3.3 SQL Statement Variables Description

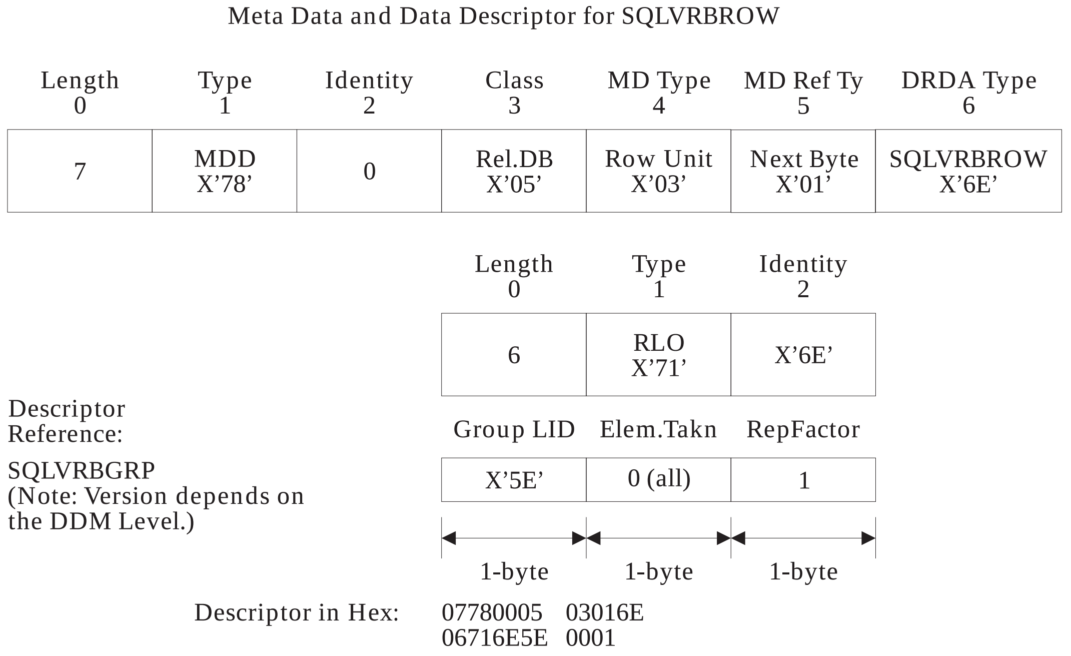


Figure 5-17 SQLVRBROW Row Descriptor (All Information for one Variable)

5.6.3.4 SQL Statement Row Description

Meta Data and Data Descriptor for SQLSTT

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X’78’	0	Rel.DB X’05’	Row Unit X’03’	Next Byte X’01’	SQLSTT X’6C’

Length 0	Type 1	Identity 2
6	RLO X’71’	X’6C’

Descriptor Reference:

SQLSTTGRP

Group LID	Elem.Takn	RepFactor
X’5C’	0 (all)	1
1-byte	1-byte	1-byte

Descriptor in Hex:

```
07780005   03016C
06716C5C  0001
```

Figure 5-18 SQLSTT Row Descriptor (One SQL Statement)

This row descriptor consists of a single long variable character string that contains a full SQL statement. The statement begins with the first character of the SQL verb (such as U for UPDATE) and ends with the last non-blank character before any terminating punctuation.

The binder must specially treat SQL statements that contain references to program variables. Detailed rules are listed in Section 7.10 on page 296.

Valid statements are defined in *ISO/IEC 9075: 1992, Database Language SQL* (hereafter abbreviated to *ISO SQL*). Product-specific non-ISO SQL statements are described in the individual product references.

5.6.3.5 SQL Object Name Row Description

Meta Data and Data Descriptor for SQLOBJNAM

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel;DB X'05'	Row Unit X'03'	Next Byte X'01'	SQLOBJNAM X'6A'

Length 0	Type 1	Identity 2
6	RLO X'71'	X'6A'

Descriptor Reference:

SQLOBJGRP

Group LID	Elem.Takn	RepFactor
X'5A'	0 (all)	1
1-byte	1-byte	1-byte

```
Descriptor in Hex:   07780005   03016A
                     06716A5A  0001
```

Figure 5-19 SQLOBJNAM Row Descriptor (One SQL Object Name)

5.6.3.6 SQL Number of Elements Row Description

Meta Data and Data Descriptor for SQLNUMROW

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel;DB X'05'	Row Unit X'03'	Next Byte X'01'	SQLNUMROW X'68'

Length 0	Type 1	Identity 2
6	RLO X'71'	X'68'

Descriptor Reference:

SQLNUMGRP

Group LID	Elem.Takn	RepFactor
X'58'	0 (all)	1
1-byte	1-byte	1-byte

```
Descriptor in Hex:   07780005   030168
                     06716858   0001
```

Figure 5-20 SQLNUMROW Row Descriptor

5.6.3.7 SQL Privileges Area Repeating Group Row Description

Meta Data and Data Descriptor for SQLPAROW

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Row Unit X'03'	Next Byte X'01'	SQLPAROW X'66'

Descriptor Reference:

SQLPAGRP

Length 0	Type 1	Identity 2
6	RLO X'71'	X'66'
Group LID	**Elem.Takn**	**RepFactor**
X'56'	0 (all)	1
1-byte	1-byte	1-byte

Descriptor in Hex: 07780005 030166
06716656 0001

Figure 5-21 SQLPAROW SQLPA Repeating Group Row Descriptor

5.6.3.8 SQL Communication Area Row Description

Meta Data and Data Descriptor for SQLCARD

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Row Unit X'03'	Next Byte X'01'	SQLCARD X'64'

Length 0	Type 1	Identity 2
6	RLO X'71'	X'64'

Descriptor Reference:

SQLCAGRP

Group LID	Elem.Takn	RepFactor
X'54'	0 (all)	1
1-byte	1-byte	1-byte

Descriptor in Hex: 07780005 030164
06716454 0001

Figure 5-22 SQLCARD Row Descriptor (SQLCA as Presented on the Link)

5.6.3.9 SQL Data Area Row Description

Meta Data and Data Descriptor for SQLDAROW

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Row Unit X'03'	Next Byte X'01'	SQLDAROW X'60'

Length 0	Type 1	Identity 2
6	RLO X'71'	X'60'

Descriptor Reference:

SQLDAGRP (Note: Version depends on the DDM Level.)

Group LID	Elem.Takn	RepFactor
X'50'	0 (all)	1
1-byte	1-byte	1-byte

Descriptor in Hex:

```
07780005   030160
06716050   0001
```

Figure 5-23 SQLDAROW Row Descriptor (SQLDA Describing Variables during Bind)

5.6.4 Early Group Descriptors

Figure 5-24 to Figure 5-34 on page 192 describe DRDA-defined Early Group Descriptors. These define database management system understood groups of fields. As with the late descriptors, length attributes, sequence, and collection nullability are specified.

5.6.4.1 SQL Result Set Group Description

Meta Data and Data Descriptor for SQLRSGRP

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	GroupUnit X'02'	Next Byte X'01'	SQLRSGRP X'5F'

Length 0	Type 1	Identity 2
15	GDA X'75'	X'5F'

Data Label:	DRDA Type:	Env. LID (1-byte)	Length Override (2-bytes)
SQLRSLOCATOR	RSL	X'14'	4
SQLRSNAME_m	VCM	X'3E'	30
SQLRSNAME_s	VCS	X'32'	30
SQLRSNUMROWS	I4	X'02'	4

```
Descriptor in Hex:  07780005   02015F
                    0F755F14   00043E00  1E32001E  020004
```

Figure 5-24 SQLRSGRP Group Descriptor (Information for One Result Set)

Table 5-4 on page 178 describes the usage of each of the fields of the SQL Result Set group.

Table 5-4 SQL Result Set Field Usage

Field Name	Usage
SQLRSLOCATOR	Result set locator value. The value of this field should be unique within the final SQL Result array.
SQLRSNAME_m SQLRSNAME_s	Name of the result set as provided by the stored procedure that generated the result set. This string can have any syntax that the application requester can handle. The value of this field should be unique within the final SQL Result array. SQLRSNAME_m and SQLRSNAME_s are mutually exclusive; that is, only one can be specified with a non-zero length. If both are non-zero, return DTAMCHRM.
SQLRSNUMROWS	The number of rows (or estimated number of rows) in the result set.

Note: All fields above are required.

5.6.4.2 SQL Result Set Column Information Group Description

Meta Data and Data Descriptor for SQLCIGRP

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	GroupUnit X'02'	Next Byte X'01'	SQLCIGRP X'5B'

Length 0	Type 1	Identity 2
21	GDA X'75'	X'5B'

Data Label:	DRDA Type:	Env. LID	Length Override
SQLCNAME_m	VCM	X'3E'	30
SQLCNAME_s	VCS	X'32'	30
SQLCLABEL_m	VCM	X'3E'	30
SQLCLABEL_s	VCS	X'32'	30
SQLCOMMENTS_m	VCM	X'3E'	254
SQLCOMMENTS_s	VCS	X'32'	254
		1-byte	2-bytes

```
Descriptor in Hex:  07780005  02015B
                    15755B3E  001E3200  1E3E001E  32001E3E
                    00FE3200  FE
```

Figure 5-25 SQLCIGRP Group Descriptor (Information for One Column)

Table 5-5 on page 180 describes the usage of each of the fields of the SQL Result Set Column Information group.

Table 5-5 SQL Result Set Column Information Field Usage

Field Name	Usage
SQLCNAME_m SQLCNAME_s	Name of the column of a result set returned by a stored procedure as it would appear in an SQL statement. SQLCNAME_m and SQLCNAME_s are mutually exclusive; that is, only one can be specified with a non-zero length. If both are non-zero, then process as if DTAMCHRM had been received.
SQLCLABEL_m SQLCLABEL_s	Descriptive label associated with this column. SQLCLABEL_m and SQLCLABEL_s are mutually exclusive; that is, only one can be specified with a non-zero length. If both are non-zero, then process as if DTAMCHRM had been received.
SQLCCOMMENTS_m SQLCCOMMENTS_s	Comments or remarks (long description) associated with this column. SQLCCOMMENTS_m and SQLCCOMMENTS_s are mutually exclusive; that is, only one can be specified with a non-zero length. If both are non-zero, then process as if DTAMCHRM had been received.

Note: The fields above are not required. When names, labels, or comments are unavailable, two zero-length strings (four bytes containing X'00000000') are returned for each type of information that is unavailable.

5.6.4.3 SQL Statement Variables Group Description (DDM Levels Below 6)

Meta Data and Data Descriptor for SQLVRBGRP

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	GroupUnit X'02'	Next Byte X'01'	SQLVRBGRP X'5E'

Length 0	Type 1	Identity 2
30	GDA X'75'	X'5E'

Data Label:	DRDA Type:	Env. LID	Length Override
SQLPRECISION	I2	X'04'	2
SQLSCALE	I2	X'04'	2
SQLLENGTH	I4	X'02'	4
SQLTYPE	I2	X'04'	2
SQLCCSID	FB	X'26'	2
SQLNAME_m	VCM	X'3E'	64
SQLNAME_s	VCS	X'32'	64
SQLDIAGNAME_m	VCM	X'3E'	64
SQLDIAGNAME_s	VCS	X'32'	64
		1-byte	2-bytes

Descriptor in Hex:

```
07780005  02015E
1E755E04  00020400  02020004  04000226
00023E00  40320040  3E004032  0040
```

The abbreviations _m and _s stand for mixed and single, respectively.

Figure 5-26 SQLVRBGRP Group Descriptor (Info. for One Variable) (DDM Levels Below 6)

Table 5-6 on page 182 describes the usage of each of the fields of the DRDA SQL Data Area.

Table 5-6 DRDA SQL Data Area Field Usage (DDM Levels Below 6)

Field Name	Usage
SQLPRECISION	Precision of a fixed decimal field—0 for other types.
SQLSCALE	Scale of a fixed decimal or zoned decimal field—0 for other types.
SQLLENGTH	Length of the field—not counting the length field.
SQLTYPE	SQL Data Type associated with this field.
SQLCCSID	0 or the CCSID for this column.
SQLNAME_m SQLNAME_s	Name of the program variable as it appeared in the original SQL statement. This string can have any syntax that the application requester can handle. The same name can be used several times when structure expansions are performed. SQLNAME_m and SQLNAME_s are mutually exclusive; that is, only one can be specified with a non-zero length. If both are non-zero, return DTAMCHRM.
SQLDIAGNAME_m SQLDIAGNAME_s	Some fully qualified name of a program variable. This string can have any syntax that the application requester can handle. When the values in this field are identical for different rows in the final SQL Statement Variables Array, they refer to the same program variable instance. A length of zero specifies the default. The default value for this field is the value of the related SQLNAME. SQLDIAGNAME_m and SQLDIAGNAME_s are mutually exclusive; that is, only one can be specified with a non-zero length. If both are non-zero, return DTAMCHRM.

Note: All fields above are required.

5.6.4.4 SQL Statement Variables Group Description (DDM Level 6 and Above)

Meta Data and Data Descriptor for SQLVRBGRP

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X’78’	0	Rel.DB X’05’	GroupUnit X’02’	Next Byte X’01’	SQLVRBGRP X’5E’

Length 0	Type 1	Identity 2
30	GDA X’75’	X’5E’

Data Label:	DRDA Type:	Env. LID	Length Override
SQLPRECISION	I2	X’04’	2
SQLSCALE	I2	X’04’	2
SQLLENGTH	I8	X’16’	8
SQLTYPE	I2	X’04’	2
SQLCCSID	FB	X’26’	2
SQLNAME_m	VCM	X’3E’	64
SQLNAME_s	VCS	X’32’	64
SQLDIAGNAME_m	VCM	X’3E’	64
SQLDIAGNAME_s	VCS	X’32’	64
		1-byte	2-bytes

```
Descriptor in Hex:   07780005  02015E
                     1E755E04  00020400  02160008  04000226
                     00023E00  40320040  3E004032  0040
```

The abbreviations _m and _s stand for mixed and single, respectively.

Figure 5-27 SQLVRBGRP Group Descriptor (Info. for One Variable) (DDM Level 6 and Above)

Table 5-7 on page 184 describes the usage of each of the fields of the DRDA SQL Data Area.

Table 5-7 DRDA SQL Data Area Field Usage (DDM Level 6 and Above)

Field Name	Usage
SQLPRECISION	Precision of a fixed decimal field—0 for other types.
SQLSCALE	Scale of a fixed decimal or zoned decimal field—0 for other types.
SQLLENGTH	Length of the field—not counting the length field.
SQLTYPE	SQL Data Type associated with this field.
SQLCCSID	0 or the CCSID for this column.
SQLNAME_m SQLNAME_s	Name of the program variable as it appeared in the original SQL statement. This string can have any syntax that the application requester can handle. The same name can be used several times when structure expansions are performed. SQLNAME_m and SQLNAME_s are mutually exclusive; that is, only one can be specified with a non-zero length. If both are non-zero, return DTAMCHRM.
SQLDIAGNAME_m SQLDIAGNAME_s	Some fully qualified name of a program variable. This string can have any syntax that the application requester can handle. When the values in this field are identical for different rows in the final SQL Statement Variables Array, they refer to the same program variable instance. A length of zero specifies the default. The default value for this field is the value of the related SQLNAME. SQLDIAGNAME_m and SQLDIAGNAME_s are mutually exclusive; that is, only one can be specified with a non-zero length. If both are non-zero, return DTAMCHRM.

Note: All fields above are required.

5.6.4.5 SQL Statement Group Description

Meta Data and Data Descriptor for SQLSTTGRP

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X’78’	0	Rel.DB X’05’	GroupUnit X’02’	Next Byte X’01’	SQLSTTGRP X’5C’

Length 0	Type 1	Identity 2
9	GDA X’75’	X’5C’

Data Label:	DRDA Type:	Env. LID	Length Override
SQLSTATEMENT_m	LVCM	X’40’	32767
SQLSTATEMENT_s	LVCS	X’34’	32767
		1-byte	2-bytes

Descriptor in Hex: 07780005 02015C
09755C40 7FFF347F FF

Figure 5-28 SQLSTTGRP Group Descriptor (One SQL Statement)

This group defines a pair of variable character strings, one of which contains an SQL statement. SQLSTATEMENT_m and SQLSTATEMENT_s are mutually exclusive; that is, only one non-zero length value can be specified for the duplicated field SQLSTATEMENT. If both are non-zero, return DTAMCHRM.

5.6.4.6 SQL Object Name Group Description

Meta Data and Data Descriptor for SQLOBJGRP

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	GroupUnit X'02'	Next Byte X'01'	SQLOBJGRP X'5A'

Length 0	Type 1	Identity 2
9	GDA X'75'	X'5A'

Data Label:	DRDA Type:	Env. LID	Length Override
SQLOBJECTNAME_m	VCM	X'3E'	254
SQLOBJECTNAME_s	VCS	X'32'	254
		1-byte	2-bytes

Descriptor in Hex:

```
07780005   02015A
09755A3E  00FE3200  FE
```

Figure 5-29 SQLOBJGRP Group Descriptor (One SQL Object Name)

This group defines a pair of variable character strings, one of which contains the name of a collection, package, index, table, or view. The name can be a one, two, or three-part relational database object name. SQLOBJECTNAME_m and SQLOBJECTNAME_s are mutually exclusive; that is, only one non-zero length value can be specified for the duplicated field. If both are non-zero, return DTAMCHRM.

5.6.4.7 SQL Number of Elements Group Description

Meta Data and Data Descriptor for SQLNUMGRP

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	GroupUnit X'02'	Next Byte X'01'	SQLNUMGRP X'58'

Length 0	Type 1	Identity 2
6	GDA X'75'	X'58'

Data Label:	DRDA Type:	Env. LID	Length Override
SQLNUM	12	X'04'	2
		1-byte	2-bytes

Descriptor in Hex: 07780005 020158
06755804 0002

Figure 5-30 SQLNUMGRP Group Descriptor

This group defines the number of entries in some DRDA array objects. It is used to allocate internal storage for the object before the entire object is received.

5.6.4.8 SQL Privileges Area Group Description

Meta Data and Data Descriptors for SQLPAGRP

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	GroupUnit X'02'	Next Byte X'01'	SQLPAGRP X'56'

Length 0	Type 1	Identity 2
27	GDA X'75'	X'56'

Data Label:	DRDA Type:	Env. LID	Length Override
SQLPCOL_m	VCM	X'3E'	18
SQLPCOL_s	VCS	X'32'	18
SQLPOBJ_m	VCM	X'3E'	18
SQLPOBJ_s	VCS	X'32'	18
SQLPALS	FCS	X'30'	1
SQLPOTP	FCS	X'30'	1
SQLPOWN	FCS	X'30'	1
SQLPPRVS	FCS	X'30'	25

1-byte 2-bytes

```
Descriptor in Hex:  07780005  020156
                    1B75563E  00123200  123E0012  32001230
                    00013000  01300001  300019
```

Figure 5-31 SQLPAGRP Group Descriptor

Table 5-8 on page 189 describes the usage of each of the fields of the SQL Privileges Area Repeating data area.

Table 5-8 DRDA SQL Privileges Area Repeating Field Usage

Field Name	Usage
SQLPCOL_m SQLPCOL_s	Collection-ID of the object being described. SQLPCOL_m and SQLPCOL_s are mutually exclusive; that is, only one can be specified with a non-zero length. If both are non-zero, then process as if DTAMCHRM had been received.
SQLPOBJ_m SQLPOBJ_s	Name of an object being described. SQLPOBJ_m and SQLPOBJ_s are mutually exclusive; that is, only one can be specified with a non-zero length. If both are non-zero, then process as if DTAMCHRM had been received.
SQLPALS	Specifies whether SQLPOBJ is an alias.
SQLPOTP	Object type.
SQLPOWN	Specifies whether the user owns or has special authority on the object.
SQLPPRVS	String describing the privileges held by the executor of the DESCRIBE PRIVILEGES command on the object being described.

SQL and the individual implementations of SQL define meanings of the values in each of the fields in Table 5-8. See *ISO SQL* and specific product references for details.

5.6.4.9 SQL Communication Area Group Description

Meta Data and Data Descriptors for SQLCAGRP

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	GroupUnit X'02'	Next Byte X'01'	SQLCAGRP X'54'

Length 0	Type 1	Identity 2
15	N-GDA X'76'	X'54'

Reference or Data Label:	DRDA Type:	Env. LID	Length Override
SQLCODE	I4	X'02'	4
SQLSTATE	FCS	X'30'	5
SQLERRPROC	FCS	X'30'	8
SQLCAXGRP	N / A	X'52''	0

1-byte 2-bytes

```
Descriptor in Hex:  07780005   020154
                    0F765402   00043000  05300008  520000
```

Figure 5-32 SQLCAGRP Group Descriptor (Nullable)

SQL and individual implementations define the semantics of the values of SQLCODE and SQLSTATE.

The values default to 0 or the normal or non-error condition. Therefore, a null SQLCA indicates everything is fine: SQLSTATE='00000'. See *ISO SQL* and specific product references for details. See also Chapter 8 on page 331.

5.6.4.10 SQL Communication Area Exceptions Group Description

Meta Data and Data Descriptors for SQLCAXGRP

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	GroupUnit X'02'	Next Byte X'01'	SQLCAXGRP X'52'

Length 0	Type 1	Identity 2
63	N-GDA X'76'	X'52'

Data Label:	DRDA Type:	Env. LID	Length Override
SQLRDBNME	FCS	X'30'	18
SQLERRD1	I4	X'02'	4
SQLERRD2	I4	X'02'	4
SQLERRD3	I4	X'02'	4
SQLERRD4	I4	X'02'	4
SQLERRD5	I4	X'02'	4
SQLERRD6	I4	X'02'	4
SQLWARN0	FCS	X'30'	1
SQLWARN1	FCS	X'30'	1
SQLWARN2	FCS	X'30'	1
SQLWARN3	FCS	X'30'	1
SQLWARN4	FCS	X'30'	1
SQLWARN5	FCS	X'30'	1
SQLWARN6	FCS	X'30'	1
SQLWARN7	FCS	X'30'	1
SQLWARN8	FCS	X'30'	1
SQLWARN9	FCS	X'30'	1
SQLWARNA	FCS	X'30'	1
SQLERRMSG_m	VCM	X'3E'	70
SQLERRMSG_s	VCS	X'32'	70
		1-byte	2-bytes

```
Descriptor in Hex:  07780005  020152
                    3F765230  00120200  04020004  02000402
                    00040200  04020004  30000130  00013000
                    01300001  30000130  00013000  01300001
                    30000130  00013000  013E0046  320046
```

Figure 5-33 SQLCAXGRP Group Descriptor (Nullable)

SQL and individual implementations define the semantics of the values in each of the fields in Figure 5-33. All fields default to normal or non-error condition. A null SQLCA indicates everything is fine. See *ISO SQL* and product references for details.

SQLERRMSG_m and SQLERRMSG_s are mutually exclusive; that is, only one non-zero length can be specified for the field SQLERRMSG. If both are non-zero, then process as if DTAMCHRM had been received.

5.6.4.11 SQL Data Area Group Description (DDM Levels Below 6)

Meta Data and Data Descriptor for SQLDAGRP

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	GroupUnit X'02'	Next Byte X'01'	SQLDAGRP X'50'

Length 0	Type 1	Identity 2
36	GDA X'75'	X'50'

Data Label:	DRDA Type:	Env. LID	Length Override
SQLPRECISION	I2	X'04'	2
SQLSCALE	I2	X'04'	2
SQLLENGTH	I4	X'02'	4
SQLTYPE	I2	X'04'	2
SQLCCSID	FB	X'26'	2
SQLNAME_m	VCM	X'3E'	30
SQLNAME_s	VCS	X'32'	30
SQLLABEL_m	VCM	X'3E'	30
SQLLABEL_s	VCS	X'32'	30
SQLCOMMENTS_m	VCM	X'3E'	254
SQLCOMMENTS_s	VCS	X'32'	254

$\xleftrightarrow{\text{1-byte}}$ $\xleftrightarrow{\text{2-bytes}}$

```
Descriptor in Hex:  07780005  020150
                    27755004  00020400  02020004  04000226
                    00023E00  1E32001E  3E001E32  001E3E00
                    FE3200FE
```

Figure 5-34 SQLDAGRP Group Descriptor (Info. for One Column) (DDM Levels Below 6)

Table 5-9 on page 193 describes the usage of each of the fields of the DRDA SQL Data Area.

Table 5-9 DRDA SQL Data Area Field Usage (DDM Levels Below 6)

Field Name	Usage
SQLPRECISION	Precision of a fixed decimal field—0 for other types.
SQLSCALE	Scale of a fixed decimal or zoned decimal field—0 for other types.
SQLLENGTH	Length of the field not counting the length field.
SQLTYPE	SQL Data Type associated with this field.
SQLCCSID	0 or the CCSID for this column.
SQLNAME_m SQLNAME_s	Name of the column as it would appear in an SQL statement. This field, at times, contains host variable names or the derivation expression for derived columns (Col1+Col2). SQLNAME_m and SQLNAME_s are mutually exclusive; that is, only one can be specified with a non-zero length. If both are non-zero, then process as if DTAMCHRM had been received.
SQLLABEL_m SQLLABEL_s	Descriptive label associated with this column. SQLLABEL_m and SQLLABEL_s are mutually exclusive; that is, only one can be specified with a non-zero length. If both are non-zero, then process as if DTAMCHRM had been received.
SQLCOMMENTS_m SQLCOMMENTS_s	Comments or remarks (long description) associated with this column. SQLCOMMENTS_m and SQLCOMMENTS_s are mutually exclusive; that is, only one can be specified with a non-zero length. If both are non-zero, then process as if DTAMCHRM had been received.

DRDA requires each field of the SQL Data Area mentioned in Table 5-9. If a value is available through DESCRIBE at the machine that is constructing this SQL Data Area, the relational database manager at the application server must provide it and send it to the other end. When SQLNAME, SQLLABEL, or SQLCOMMENTS is unavailable, two zero-length strings (four bytes containing X‘00000000’) are returned for each.

5.6.4.12 SQL Data Area Group Description (DDM Level 6 and Above)

Meta Data and Data Descriptor for SQLDAGRP

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	GroupUnit X'02'	Next Byte X'01'	SQLDAGRP X'50'

Length 0	Type 1	Identity 2
36	GDA X'75'	X'50'

Data Label:	DRDA Type:	Env. LID	Length Override
SQLPRECISION	I2	X'04'	2
SQLSCALE	I2	X'04'	2
SQLLENGTH	I8	X'16'	8
SQLTYPE	I2	X'04'	2
SQLCCSID	FB	X'26'	2
SQLNAME_m	VCM	X'3E'	30
SQLNAME_s	VCS	X'32'	30
SQLLABEL_m	VCM	X'3E'	30
SQLLABEL_s	VCS	X'32'	30
SQLCOMMENTS_m	VCM	X'3E'	254
SQLCOMMENTS_s	VCS	X'32'	254
SQLUDTGRP	N / A	X'51'	0

1-byte 2-bytes

Descriptor in Hex:

```
07780005  020150
27755004  00020400  02160008  04000226
00023E00  1E32001E  3E001E32  001E3E00
FE3200FE  510000
```

Figure 5-35 SQLDAGRP Group Descriptor (Info. for One Column) (DDM Level 6 and Above)

Table 5-10 on page 195 describes the usage of each of the fields of the DRDA SQL Data Area.

Table 5-10 DRDA SQL Data Area Field Usage (DDM Level 6 and Above)

Field Name	Usage
SQLPRECISION	Precision of a fixed decimal field—0 for other types.
SQLSCALE	Scale of a fixed decimal or zoned decimal field—0 for other types.
SQLLENGTH	Length of the field not counting the length field.
SQLTYPE	SQL Data Type associated with this field.
SQLCCSID	0 or the CCSID for this column.
SQLNAME_m SQLNAME_s	Name of the column as it would appear in an SQL statement. This field, at times, contains host variable names or the derivation expression for derived columns (Col1+Col2). SQLNAME_m and SQLNAME_s are mutually exclusive; that is, only one can be specified with a non-zero length. If both are non-zero, then process as if DTAMCHRM had been received.
SQLLABEL_m SQLLABEL_s	Descriptive label associated with this column. SQLLABEL_m and SQLLABEL_s are mutually exclusive; that is, only one can be specified with a non-zero length. If both are non-zero, then process as if DTAMCHRM had been received.
SQLCOMMENTS_m SQLCOMMENTS_s	Comments or remarks (long description) associated with this column. SQLCOMMENTS_m and SQLCOMMENTS_s are mutually exclusive; that is, only one can be specified with a non-zero length. If both are non-zero, then process as if DTAMCHRM had been received.

DRDA requires each field of the SQL Data Area mentioned in Table 5-9 on page 193. If a value is available through DESCRIBE at the machine that is constructing this SQL Data Area, the relational database manager at the application server must provide it and send it to the other end. When SQLNAME, SQLLABEL, or SQLCOMMENTS is unavailable, two zero-length strings (four bytes containing X‘00000000’) are returned for each.

5.6.4.13 SQL User-Defined Data Group Description

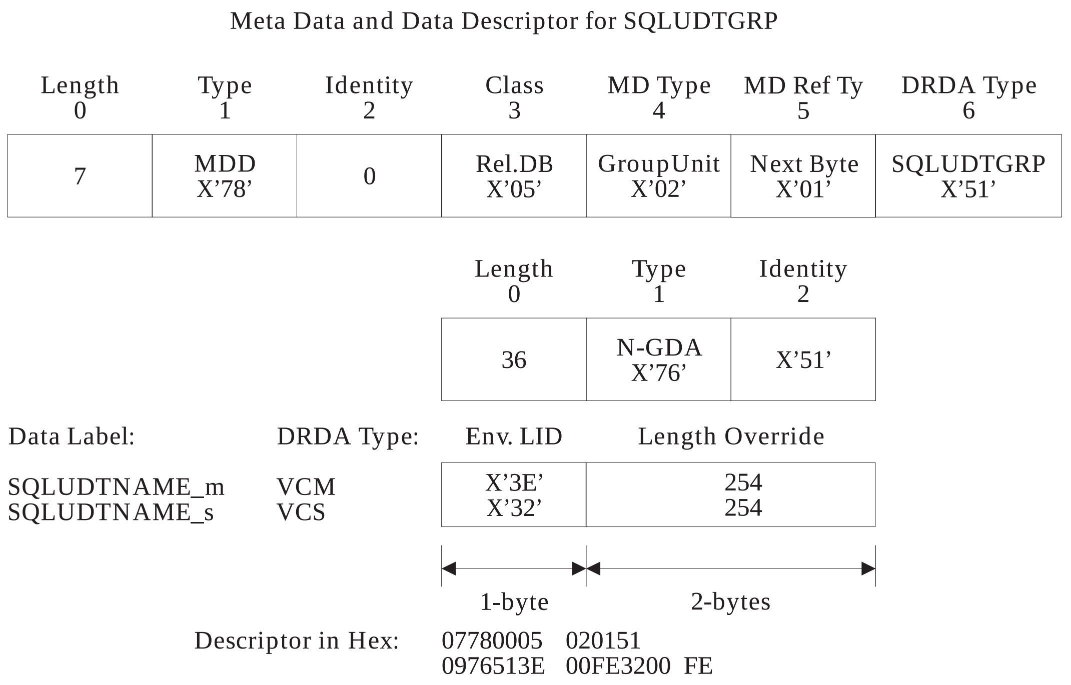


Figure 5-36 SQLUDTGRP Group Descriptor (UDT Information for One Column)

Table 5-11 describes the usage of each of the fields of the SQLUDTGRP.

Table 5-11 DRDA SQL UDT Description Field Usage"

Field Name	Usage
SQLUDTNAME_m and SQLUDTNAME_s	The name of the user-defined data type describing the data in a column. SQLUDTNAME_m and SQLUDTNAME_s are mutually exclusive; that is, only one can be specified with a non-zero length. If both are non-zero, then process as if DTAMCHRM has been received.

DRDA requires each field of the SQL UDT Group Description mentioned in Table 5-11.

5.6.5 Early Environmental Descriptors

Figure 5-37 through Figure 5-71 on page 234 show how each of the DRDA database management system environments represent each of the DRDA and SQL data types.

Figure 5-37 shows each of the parameters in the environment descriptor for one data type, namely Variable Character SBCS data. Each type consists of a Meta Data Definition (MDD) that states the DRDA semantics of the descriptor followed by a Simple Data Array (SDA) that says how that type is to be represented.

Variable Character SBCS (Example)

Length	Type	Identity	Class	MD Type	MD Ref Ty	DRDA Type
0	1	2	3	4	5	6
Meta Data (Environment-independent)						
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'32' (VCS) X'33' (NVCS)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID				Chr Siz	Mode	Fld	Length
0	1	2	3	4	5	6	7	8	9	10	11
QTDSQL370 (System / 370* Processors)											
12	X'70'	X'32'	X'11'	00000-00500(e)				1	1	32767	
12	X'70'	X'32'	X'91'	00000-00500(e)				1	1	32767	

Example Descriptor in Hex	07780005 010133
(QTDSQL370 nullable form)	0C703391 000001F4 01017FFF

(e) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.

Figure 5-37 DRDA Type X'32,33' SQL Type 448,449 Variable Character SBCS

The upper box (the meta data triplet) is identical for all machine environments. There is only one triplet defined per DRDA type. It is environment-independent.

The lower boxes in these figures are unique to each machine. The contents of these boxes are described below. These are the descriptions for the parameters shown in Figure 5-37.

Above Lower Box

The name of the environment to which this descriptor belongs. A parenthetical description follows the name. DRDA defines five environments:

- QTDSQL370 (System / 390 processors)
- QTDSQL400 (AS / 400 processors)
- QTDSQLX86 (Intel 80X86 processors)
- QTDSQLASC[38] (IEEE non-byte-reversed ASCII processors)

- QTDSQLVAX (VAX processors)

These are the names of type definitions and appear as values in the TYPDEFNAM parameter of DDM commands.

The example describes part of the 'QTDSQL370' environment.

Lower Box

Each of the lower boxes contains the SDAs for the non-nullable and nullable form for each environment specified.

Byte 0 The length of the FD:OCA SDA triplet. For DRDA, all SDAs are 12 bytes long.

Byte 1 Simple Data Array type indicator is always X‘70’ for SDAs.

Byte 2 The Local Identifier of this SDA. DRDA assigns this LID in the standard environments and directly maps the LID to the DRDA Type. The formal mapping from DRDA type to LID is through the associated FD:OCA MDD specifications. DRDA types provide a mapping path from SQL Types to FD:OCA representations.

Nullable SQL and DRDA types are all odd numbers and nullable type is one number higher than the related non-nullable type. These values are shown in hex.

For the example, two DRDA types are defined: X‘32’ corresponding to the SQL type for non-nullable Variable Character SBCS strings and X‘33’ corresponding to the nullable SQL type.

Byte 3 The FD:OCA data type indicator shows exactly how the data is represented in this environment. These values are shown in hex. For a detailed explanation of these types, see the FD:OCA Reference.

The null indicator, when defined to be present, flows as an extra byte in front of the actual data that can follow. This indicator is a one byte signed binary integer (I1) and is filled with the least significant byte that SQL returned in its indicator variable. All negative values (X‘80’-X‘FF’) represent various null data conditions. Zero indicates a complete data value follows. Positive values indicate truncation has occurred, but positive values do not occur due to DRDA’s use of natural SQLDAs. SQL, not DRDA, specifies the following values:

- 0 (X‘00’) data value follows
- −1 (X‘FF’) no data value follows
- −2 (X‘FE’) undefined result, no data value follows
- −3 to −128 (X‘FD’-X‘80’) reserved, no data value follows

In the example, FD:OCA type X‘11’ Character Variable Length represents non-nullable strings in the System 390 environment. The Nullable type uses FD:OCA type X‘91’.

38. An example of an QTDSQLASC machine is the IBM RS/6000, which has an IEEE floating point format and non-byte reversed numbers. This contrasts with the Intel floating point format that has byte-reversed floating point and integer numbers.

Bytes 4-7

The CCSID identifies the encoding of the character data. Converting the CCSID into binary form generates the four byte representation. This information is in decimal. The FD:OCA rules state that if the high order 16 bits of the CCSID field are zero, then the low order 16 bits are to be interpreted as a CCSID rather than as a code page identification. DRDA uses the CCSID format.

The CCSID is a pointer (16 bits) to a description of an encoding scheme, one or more pairs of character set and code page, and possible additional coding-related information (ACRI). See character data types in the FD:OCA Reference and CDRA Reference for information about CCSIDs.

In FD:OCA, the containing architecture is allowed to establish its own mechanisms for constructing valid FD:OCA descriptors. For DRDA, DDM provides the TYPDEFOVR parameter on the ACCRDB command as the means of establishing the CCSIDs to use for a connection. For any parameter not sent on TYPDEFOVR (such as CCSIDMBC or CCSIDDBC) no character data of any length greater than zero can flow with that type representation.

The CCSIDs shown in bytes 4 through 7 of the following figures are examples only and not part of DRDA. DRDA does not define default CCSID values. When a CCSID is required for one or more data value descriptions, either the application requester or application server must provide a Late Environmental Descriptor. When a CCSID is required for one or more data value descriptors, specify it in one of the following ways:

1. TYPEDEFOVR parameter on ACCRDB/ACCRDBRM

   This requires an MDD and an SDA. The MDD is exactly like the one for the type being specified, and the SDA is the same except that a specific CCSID is filled in. The GDA, which defines the data field characteristics, references this new SDA. (See Section 5.6.6 on page 243.)

2. TYPDEFOVR DDM command/reply

3. Late environmental descriptor

See the CDRA Reference for additional information on available CCSIDs.

Byte 8 Character Size. This field indicates the number of bytes each character takes in storage. The value 2 is used for GRAPHIC SQL Types; 1 is used for all other character, date, time, timestamp, and numeric character fields. It must be 0 for all other types.

For this example, the data is SBCS characters, so the character length is specified as 1.

Byte 9 Mode. This field is used to specify mode of interpretation of FD:OCA architecture for all variable length data types (including null terminated), such as the SBCS variable character type used in the example. The low order bit of this byte is used to control interpretation of Length Fields in SDAs for variable length types. A '0' in that bit indicates that non-zero length field values indicate the space reserved for data and that all the space is transmitted (or laid out in storage) whether or not it contains valid data. In the case of the example, the first two bytes of the data itself determine valid data length.

A ’1’ in this bit shows that non-zero length field values indicate the maximum value of the length fields that the data will contain. Only enough space to contain each data value is transmitted for each value.

The example above is a variable length field. Because DRDA does not want to transmit unnecessary bytes, Mode is set to ’1’.

Bytes 10-11

The interpretation of these bytes for the example, as well as for most other data types, is as follows:

This is the length of the field and is shown in decimal. It represents the maximum valid value. When the Group Data Array triplet overrides it, the value can be reduced. For character fields with only DBCS characters, this is the length in characters (bytes/2). For all other cases, the length is in bytes. It does not include the length of the length field (variable length types) or null indicator (nullable types).

For the example, the maximum length of data allowed is 32767 bytes. On the link, DRDA type X‘32’ could be up to 32769 bytes long and DRDA type X‘33’ up to 32770, which allows space for the length field and null indicators. The maximum value is reduced, with a Group Data Array specification, to match the actual field or column size.

Below Lower Box

Notes about values in the box. A lowercase alphabetic character identifies each note. Inside the box, all lowercase alphabetic characters are references to notes.

In the example, note (e) is referenced from the CCSID field.

The following figures show the Simple Data Arrays (SDAs) that define the representations for each DRDA type in each of the planned environments. These SDAs are bundled together logically into an environmental descriptor set for each environment. The choice of which set to use is made at ACCRDB time.

5.6.5.1 Four-Byte Integer

Length	Type	Identity	Class	MD Type	MD Ref Ty	DRDA Type
0	1	2	3	4	5	6

Meta Data (Environment-independent)

7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'02' (I4) X'03' (NI4)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	Reserved						Fld Length	
0	1	2	3	4	5	6	7	8	9	10	11

QTDSQL370 (System / 370 Processors)

12	X'70'	X'02'	X'23'	0	0	0	4
12	X'70'	X'03'	X'A3'	0	0	0	4

QTDSQL400 (AS / 400 Processors)

12	X'70'	X'02'	X'23'	0	0	0	4
12	X'70'	X'03'	X'A3'	0	0	0	4

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'02'	X'24'	0	0	0	4
12	X'70'	X'03'	X'A4'	0	0	0	4

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'02'	X'23'	0	0	0	4
12	X'70'	X'03'	X'A3'	0	0	0	4

QTDSQLVAX (VAX Processors)

12	X'70'	X'02'	X'24'	0	0	0	4
12	X'70'	X'03'	X'A4'	0	0	0	4

```
Example Descriptor in Hex     07780005  010103
(QTDSQL370 nullable form)     0C7003A3  00000000  00000004
```

Figure 5-38 DRDA Type X'02,03' SQL Type 496,497 INTEGER

The Intel Processor is the OS / 2 processor.

5.6.5.2 Two-Byte Integer

Meta Data (Environment-independent)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'04' (I2) X'05' (NI2)

QTDSQL370 (System / 370 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X'70'	X'04'	X'23'	0	0	0	2
12	X'70'	X'05'	X'A3'	0	0	0	2

QTDSQL400 (AS / 400 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X'70'	X'04'	X'23'	0	0	0	2
12	X'70'	X'05'	X'A3'	0	0	0	2

QTDSQLX86 (Intel 80X86 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X'70'	X'04'	X'24'	0	0	0	2
12	X'70'	X'05'	X'A4'	0	0	0	2

QTDSQLASC (IEEE ASCII Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X'70'	X'04'	X'23'	0	0	0	2
12	X'70'	X'05'	X'A3'	0	0	0	2

QTDSQLVAX (VAX Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X'70'	X'04'	X'24'	0	0	0	2
12	X'70'	X'05'	X'A4'	0	0	0	2

```
Example Descriptor in Hex       07780005   010105
(QTDSQL370 nullable form)       0C7005A3  00000000  00000002
```

Figure 5-39 DRDA Type X'04,05' SQL Type 500,501 SMALL INTEGER

5.6.5.3 One-Byte Integer

Length	Type	Identity	Class	MD Type	MD Ref Ty	DRDA Type
0	1	2	3	4	5	6

Meta Data (Environment-independent)

7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'06' (I1) X'07' (NI1)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	Reserved						Fld Length	
0	1	2	3	4	5	6	7	8	9	10	11

QTDSQL370 (System / 370 Processors)

12	X'70'	X'06'	X'23'	0	0	0	1
12	X'70'	X'07'	X'A3'	0	0	0	1

QTDSQL400 (AS / 400 Processors)

12	X'70'	X'06'	X'23'	0	0	0	1
12	X'70'	X'07'	X'A3'	0	0	0	1

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'06'	X'24'	0	0	0	1
12	X'70'	X'07'	X'A4'	0	0	0	1

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'06'	X'23'	0	0	0	1
12	X'70'	X'07'	X'A3'	0	0	0	1

QTDSQLVAX (VAX Processors)

12	X'70'	X'06'	X'24'	0	0	0	1
12	X'70'	X'07'	X'A4'	0	0	0	1

```
Example Descriptor in Hex     07780005  010107
(QTDSQL370 nullable form)     0C7007A3  00000000  00000001
```

Figure 5-40 DRDA Type X'06,07' SQL Type n / a,n / a

5.6.5.4 Sixteen-Byte Float

Meta Data (Environment-independent)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'08' (BF16) X'09' (NBF16)

QTDSQL370 (System / 370 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Res- erved 4 5	Bias 6	 7	Res- erved 8	Mode 9	Fld 10 Length 11
12	X'70'	X'08'	X'40'	0			0	0	16
12	X'70'	X'09'	X'C0'	0			0	0	16

QTDSQL400 (AS / 400 Processors)

0	1	2	3	4 5	6	7	8	9	10 11
12	X'70'	X'08'	X'48'	0	0	0	0	0	16
12	X'70'	X'09'	X'C8'	0	0	0	0	0	16

QTDSQLX86 (Intel 80X86 Processors)

0	1	2	3	4 5	6	7	8	9	10 11
12	X'70'	X'08'	X'47'	0	0	0	0	0	16
12	X'70'	X'09'	X'C7'	0	0	0	0	0	16

QTDSQLASC (IEEE ASCII Processors)

0	1	2	3	4 5	6	7	8	9	10 11
12	X'70'	X'08'	X'48'	0	0	0	0	0	16
12	X'70'	X'09'	X'C8'	0	0	0	0	0	16

QTDSQLVAX (VAX Processors)

0	1	2	3	4 5	6	7	8	9	10 11
12	X'70'	X'08'	X'49'	0	0	0	0	0	16
12	X'70'	X'09'	X'C9'	0	0	0	0	0	16

Example Descriptor in Hex	07780005 010109
(QTDSQL370 nullable form)	0C7009C0 00000000 00000010

Figure 5-41 DRDA Type X'08,09' SQL Type 480,481 FLOAT (16)

5.6.5.5 Eight-Byte Float

Meta Data (Environment-independent)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'0A' (BF8) X'0B' (NBF8)

QTDSQL370 (System / 370 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Res-erved 4 5	Bias 6 7	Res-erved 8	Mode 9	Fld Length 10 11
12	X'70'	X'0A'	X'40'	0		0	0	8
12	X'70'	X'0B'	X'C0'	0		0	0	8

QTDSQL400 (AS / 400 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Res-erved 4 5	Bias 6	7	Res-erved 8	Mode 9	Fld Length 10 11
12	X'70'	X'0A'	X'48'	0	0	0	0	0	8
12	X'70'	X'0B'	X'C8'	0	0	0	0	0	8

QTDSQLX86 (Intel 80X86 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Res-erved 4 5	Bias 6	7	Res-erved 8	Mode 9	Fld Length 10 11
12	X'70'	X'0A'	X'47'	0	0	0	0	0	8
12	X'70'	X'0B'	X'C7'	0	0	0	0	0	8

QTDSQLASC (IEEE ASCII Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Res-erved 4 5	Bias 6	7	Res-erved 8	Mode 9	Fld Length 10 11
12	X'70'	X'0A'	X'48'	0	0	0	0	0	8
12	X'70'	X'0B'	X'C8'	0	0	0	0	0	8

QTDSQLVAX (VAX Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Res-erved 4 5	Bias 6	7	Res-erved 8	Mode 9	Fld Length 10 11
12	X'70'	X'0A'	X'49'	0	0	0	0	0	8
12	X'70'	X'0B'	X'C9'	0	0	0	0	0	8

```
Example Descriptor in Hex      07780005   01010B
(QTDSQL370 nullable form)      0C700BC0  00000000  00000008
```

Figure 5-42 DRDA Type X'0A,0B' SQL Type 480,481 FLOAT (8)

5.6.5.6 Four-Byte Float

Meta Data (Environment-independent)

Length	Type	Identity	Class	MD Type	MD Ref Ty	DRDA Type
0	1	2	3	4	5	6
7	MDD X’78’	0	Rel.DB X’05’	Data Type X’01’	Next Byte X’01’	X’0C’ (BF4) X’0D’ (NBF4)

FD Tr Ln	FD:OCA Tripl ’SDA’	FD:OCA Tripl LID	FD:OCA Field Type	Res-erved		Bias		Res-erved	Mode	Fld Length	
0	1	2	3	4	5	6	7	8	9	10	11

QTDSQL370 (System / 370 Processors)

12	X’70’	X’0C’	X’40’	0	0	0	4
12	X’70’	X’0D’	X’C0’	0	0	0	4

QTDSQL400 (AS / 400 Processors)

12	X’70’	X’0C’	X’48’	0	0	0	0	0	4
12	X’70’	X’0D’	X’C8’	0	0	0	0	0	4

QTDSQLX86 (Intel 80X86 Processors)

12	X’70’	X’0C’	X’47’	0	0	0	0	0	4
12	X’70’	X’0D’	X’C7’	0	0	0	0	0	4

QTDSQLASC (IEEE ASCII Processors)

12	X’70’	X’0C’	X’48’	0	0	0	0	0	4
12	X’70’	X’0D’	X’C8’	0	0	0	0	0	4

QTDSQLVAX (VAX Processors)

12	X’70’	X’0C’	X’49’	0	0	0	0	0	4
12	X’70’	X’0D’	X’C9’	0	0	0	0	0	4

Example Descriptor in Hex	07780005 01010D
(QTDSQL370 nullable form)	0C700DC0 00000000 00000004

Figure 5-43 DRDA Type X‘0C,0D’ SQL Type 480,481 FLOAT (4)

5.6.5.7 Fixed Decimal

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6

Meta Data (Environment-independent)

7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'0E' (FD) X'0F' (NFD)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	Reserved					Mode	Fld Length and Prec; / Scale	
0	1	2	3	4	5	6	7	8	9	10	11

QTDSQL370 (System / 370 Processors)

12 12	X'70' X'70'	X'0E' X'0F'	X'30' X'B0'	0 0	0 0	0 0	31 31	31 31

QTDSQL400 (AS / 400 Processors)

12 12	X'70' X'70'	X'0E' X'0F'	X'30' X'B0'	0 0	0 0	0 0	31 31	31 31

QTDSQLX86 (Intel 80X86 Processors)

12 12	X'70' X'70'	X'0E' X'0F'	X'30' X'B0'	0 0	0 0	0 0	31 31	31 31

QTDSQLASC (IEEE ASCII Processors)

12 12	X'70' X'70'	X'0E' X'0F'	X'30' X'B0'	0 0	0 0	0 0	31 31	31 31

QTDSQLVAX (VAX Processors)

12 12	X'70' X'70'	X'0E' X'0F'	X'30' X'B0'	0 0	0 0	0 0	31 31	31 31

```
Example Descriptor in Hex      07780005   01010F
(QTDSQL370 nullable form)      0C700FB0  00000000  00001F1F
```

Figure 5-44 DRDA Type X'0E,0F' SQL Type 484,485 FIXED DECIMAL

5.6.5.8 Zoned Decimal

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
Meta Data (Environment-independent)						
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'10' (ZD) X'11' (NZD)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	Mode 9	Fld Length and Prec; / Scale 10	11
QTDSQL370 (System / 370 Processors)								
12	X'70'	X'10'	X'33'	0	0	0	31	31
12	X'70'	X'11'	X'B3'	0	0	0	31	31
QTDSQL400 (AS / 400 Processors)								
12	X'70'	X'10'	X'33'	0	0	0	31	31
12	X'70'	X'11'	X'B3'	0	0	0	31	31
QTDSQLX86 (Intel 80X86 Processors)								
12	X'70'	X'10'	X'35'	0	0	0	31	31
12	X'70'	X'11'	X'B5'	0	0	0	31	31
QTDSQLASC (IEEE ASCII Processors)								
12	X'70'	X'10'	X'35'	0	0	0	31	31
12	X'70'	X'11'	X'B5'	0	0	0	31	31
QTDSQLVAX (VAX Processors)								
12	X'70'	X'10'	X'35'	0	0	0	31	31
12	X'70'	X'11'	X'B5'	0	0	0	31	31

```
Example Descriptor in Hex      07780005  010111
(QTDSQL370 nullable form)      0C7011B3  00000000  00001F1F
```

Figure 5-45 DRDA Type X'10,11' SQL Type 488,489 ZONED DECIMAL

5.6.5.9 Numeric Character

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6

Meta Data (Environment-independent)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'12' (N) X'13' (NN)

QTDSQL370 (System / 370 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	8	Mode 9	Fld Length and Prec; / Scale 10	11
12	X'70'	X'12'	X'32'	00000-00500(e)	1	0	31	31
12	X'70'	X'13'	X'B2'	00000-00500(e)	1	0	31	31

QTDSQL400 (AS / 400 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	8	Mode 9	Fld Length and Prec; / Scale 10	11
12	X'70'	X'12'	X'32'	00000-00500(e)	1	0	31	31
12	X'70'	X'13'	X'B2'	00000-00500(e)	1	0	31	31

QTDSQLX86 (Intel 80X86 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	8	Mode 9	Fld Length and Prec; / Scale 10	11
12	X'70'	X'12'	X'32'	00000-00850(e)	1	0	31	31
12	X'70'	X'13'	X'B2'	00000-00850(e)	1	0	31	31

QTDSQLASC (IEEE ASCII Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	8	Mode 9	Fld Length and Prec; / Scale 10	11
12	X'70'	X'12'	X'32'	00000-00819(e)	1	0	31	31
12	X'70'	X'13'	X'B2'	00000-00819(e)	1	0	31	31

QTDSQLVAX (VAX Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	8	Mode 9	Fld Length and Prec; / Scale 10	11
12	X'70'	X'12'	X'32'	00000-00819(e)	1	0	31	31
12	X'70'	X'13'	X'B2'	00000-00819(e)	1	0	31	31

```
Example Descriptor in Hex      07780005   010113
(QTDSQL370 nullable form)      0C7013B2   000001F4  01001F1F
```

(e) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.

Figure 5-46 DRDA Type X'12,13' SQL Type 504,505 NUMERIC CHARACTER

5.6.5.10 Result Set Locator

Meta Data (Environment-independent)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X’78’	0	Rel.DB X’05’	Data Type X’01’	Next Byte X’01’	X’14’ (RSL) X’15’ (NRSL)

QTDSQL370 (System / 370 Processors)

FD Tr Ln 0	FD:OCA Tripl ’SDA’ 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X’70’	X’14’	X’23’	0	0	0	4
12	X’70’	X’15’	X’A3’	0	0	0	4

QTDSQL400 (AS / 400 Processors)

FD Tr Ln 0	FD:OCA Tripl ’SDA’ 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X’70’	X’14’	X’23’	0	0	0	4
12	X’70’	X’15’	X’A3’	0	0	0	4

QTDSQLX86 (Intel 80X86 Processors)

FD Tr Ln 0	FD:OCA Tripl ’SDA’ 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X’70’	X’14’	X’24’	0	0	0	4
12	X’70’	X’15’	X’A4’	0	0	0	4

QTDSQLASC (IEEE ASCII Processors)

FD Tr Ln 0	FD:OCA Tripl ’SDA’ 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X’70’	X’14’	X’23’	0	0	0	4
12	X’70’	X’15’	X’A3’	0	0	0	4

QTDSQLVAX (VAX Processors)

FD Tr Ln 0	FD:OCA Tripl ’SDA’ 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X’70’	X’14’	X’24’	0	0	0	4
12	X’70’	X’15’	X’A4’	0	0	0	4

Example Descriptor in Hex	07780005 010115
(QTDSQL370 nullable form)	0C7015A3 00000000 00000004

Figure 5-47 DRDA Type X‘14,15’ SQL Type 972,973 RESULT SET LOCATOR

5.6.5.11 Eight-Byte Integer

Length	Type	Identity	Class	MD Type	MD Ref Ty	DRDA Type
0	1	2	3	4	5	6

Meta Data (Environment-independent)

7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'16' (I8) X'17' (NI8)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	Reserved						Fld	Length
0	1	2	3	4	5	6	7	8	9	10	11

QTDSQL370 (System / 370 Processors)

12 12	X'70' X'70'	X'16' X'17'	X'23' X'A3'	0 0	0 0	0 0	0 0	8 8

QTDSQL400 (AS / 400 Processors)

12 12	X'70' X'70'	X'16' X'17'	X'23' X'A3'	0 0	0 0	0 0	0 0	8 8

QTDSQLX86 (Intel 80X86 Processors)

12 12	X'70' X'70'	X'16' X'17'	X'24' X'A4'	0 0	0 0	0 0	0 0	8 8

QTDSQLASC (IEEE ASCII Processors)

12 12	X'70' X'70'	X'16' X'17'	X'23' X'A3'	0 0	0 0	0 0	0 0	8 8

QTDSQLVAX (VAX Processors)

12 12	X'70' X'70'	X'16' X'17'	X'24' X'A4'	0 0	0 0	0 0	0 0	8 8

```
Example Descriptor in Hex      07780005   010117
(QTDSQL370 nullable form)      0C7017A3  00000000  00000008
```

Figure 5-48 DRDA Type X'16,17' SQL Type 492,493 EIGHT-BYTE INTEGER

5.6.5.12 Large Object Bytes Locator

Meta Data (Environment-independent)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'18' (OBL) X'19' (NOBL)

QTDSQL370 (System / 370 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X'70'	X'18'	X'01'	0	0	0	4
12	X'70'	X'19'	X'81'	0	0	0	4

QTDSQL400 (AS / 400 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X'70'	X'18'	X'01'	0	0	0	4
12	X'70'	X'19'	X'81'	0	0	0	4

QTDSQLX86 (Intel 80X86 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X'70'	X'18'	X'01'	0	0	0	4
12	X'70'	X'19'	X'81'	0	0	0	4

QTDSQLASC (IEEE ASCII Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X'70'	X'18'	X'01'	0	0	0	4
12	X'70'	X'19'	X'81'	0	0	0	4

QTDSQLVAX (VAX Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X'70'	X'18'	X'01'	0	0	0	4
12	X'70'	X'19'	X'81'	0	0	0	4

```
Example Descriptor in Hex      07780005   010119
(QTDSQL370 nullable form)      0C701981   00000000  00000004
```

Figure 5-49 DRDA Type X'18,19' SQL Type 960,961 LARGE OBJECT BYTES LOCATOR

5.6.5.13 Large Object Character Locator

Length	Type	Identity	Class	MD Type	MD Ref Ty	DRDA Type
0	1	2	3	4	5	6

Meta Data (Environment-independent)

7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'1A' (OCL) X'1B' (NOCL)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	Reserved						Fld	Length
0	1	2	3	4	5	6	7	8	9	10	11

QTDSQL370 (System / 370 Processors)

12	X'70'	X'1A'	X'01'	0	0	0	4
12	X'70'	X'1B'	X'81'	0	0	0	4

QTDSQL400 (AS / 400 Processors)

12	X'70'	X'1A'	X'01'	0	0	0	4
12	X'70'	X'1B'	X'81'	0	0	0	4

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'1A'	X'01'	0	0	0	4
12	X'70'	X'1B'	X'81'	0	0	0	4

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'1A'	X'01'	0	0	0	4
12	X'70'	X'1B'	X'81'	0	0	0	4

QTDSQLVAX (VAX Processors)

12	X'70'	X'1A'	X'01'	0	0	0	4
12	X'70'	X'1B'	X'81'	0	0	0	4

```
Example Descriptor in Hex      07780005   01011B
(QTDSQL370 nullable form)      0C701B81  00000000  00000004
```

Figure 5-50 DRDA Type X'1A,1B' SQL Type 964,965 LARGE OBJ. CHAR. SBCS LOCATOR

5.6.5.14 Large Object Character DBCS Locator

Meta Data (Environment-independent)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'1C' (OCDL) X'1D' (NOCDL)

QTDSQL370 (System / 370 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12 12	X'70' X'70'	X'1C' X'1D'	X'01' X'81'	0 0	0 0	0 0	4 4

QTDSQL400 (AS / 400 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12 12	X'70' X'70'	X'1C' X'1D'	X'01' X'81'	0 0	0 0	0 0	4 4

QTDSQLX86 (Intel 80X86 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12 12	X'70' X'70'	X'1C' X'1D'	X'01' X'81'	0 0	0 0	0 0	4 4

QTDSQLASC (IEEE ASCII Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12 12	X'70' X'70'	X'1C' X'1D'	X'01' X'81'	0 0	0 0	0 0	4 4

QTDSQLVAX (VAX Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12 12	X'70' X'70'	X'1C' X'1D'	X'01' X'81'	0 0	0 0	0 0	4 4

```
Example Descriptor in Hex      07780005   01011D
(QTDSQL370 nullable form)      0C701D81  00000000  00000004
```

Figure 5-51 DRDA Type X'1C,1D' SQL Type 968,969 LARGE OBJ. CHAR. DBCS LOCATOR

5.6.5.15 Row Identifier

Meta Data (Environment-independent)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'1E' (RI) X'1F' (NRI)

QTDSQL370 (System / 370 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X'70'	X'1E'	X'02'	0	0	1	40
12	X'70'	X'1F'	X'82'	0	0	1	40

QTDSQL400 (AS / 400 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X'70'	X'1E'	X'02'	0	0	1	40
12	X'70'	X'1F'	X'82'	0	0	1	40

QTDSQLX86 (Intel 80X86 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X'70'	X'1E'	X'02'	0	0	1	40
12	X'70'	X'1F'	X'82'	0	0	1	40

QTDSQLASC (IEEE ASCII Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X'70'	X'1E'	X'02'	0	0	1	40
12	X'70'	X'1F'	X'82'	0	0	1	40

QTDSQLVAX (VAX Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X'70'	X'1E'	X'02'	0	0	1	40
12	X'70'	X'1F'	X'82'	0	0	1	40

Example Descriptor in Hex	07780005 01011F
(QTDSQL370 nullable form)	0C701F82 00000000 00010028

Figure 5-52 DRDA Type X'1E,1F' SQL Type 904,905 ROW IDENTIFIER

5.6.5.16 Date

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6

Meta Data (Environment-independent)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X’78’	0	Rel.DB X’05’	Data Type X’01’	Next Byte X’01’	X’20’ (D) X’21’ (ND)

FD Tr Ln 0	FD:OCA Tripl ’SDA’ 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Res-erved 9	Fld 10 Length 11

QTDSQL370 (System / 370 Processors)

FD Tr Ln	FD:OCA Tripl ’SDA’	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Siz	Res-erved	Fld Length
12	X’70’	X’20’	X’10’	00000-00500(e)	1	0	10
12	X’70’	X’21’	X’90’	00000-00500(e)	1	0	10

QTDSQL400 (AS / 400 Processors)

FD Tr Ln	FD:OCA Tripl ’SDA’	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Siz	Res-erved	Fld Length
12	X’70’	X’20’	X’10’	00000-00500(e)	1	0	10
12	X’70’	X’21’	X’90’	00000-00500(e)	1	0	10

QTDSQLX86 (Intel 80X86 Processors)

FD Tr Ln	FD:OCA Tripl ’SDA’	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Siz	Res-erved	Fld Length
12	X’70’	X’20’	X’10’	00000-00850(e)	1	0	10
12	X’70’	X’21’	X’90’	00000-00850(e)	1	0	10

QTDSQLASC (IEEE ASCII Processors)

FD Tr Ln	FD:OCA Tripl ’SDA’	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Siz	Res-erved	Fld Length
12	X’70’	X’20’	X’10’	00000-00819(e)	1	0	10
12	X’70’	X’21’	X’90’	00000-00819(e)	1	0	10

QTDSQLVAX (VAX Processors)

FD Tr Ln	FD:OCA Tripl ’SDA’	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Siz	Res-erved	Fld Length
12	X’70’	X’20’	X’10’	00000-00819(e)	1	0	10
12	X’70’	X’21’	X’90’	00000-00819(e)	1	0	10

```
Example Descriptor in Hex      07780005   010121
(QTDSQL370 nullable form)      0C702190   000001F4  0100000A
```

(e) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.

Figure 5-53 DRDA Type X‘20,21’ SQL Type 384,385 DATE

5.6.5.17 Time

Meta Data (Environment-independent)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'22' (T) X'23' (NT)

QTDSQL370 (System / 370 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Res-erved 9	Fld Length 10 11
12	X'70'	X'22'	X'10'	00000-00500(e)	1	0	8
12	X'70'	X'23'	X'90'	00000-00500(e)	1	0	8

QTDSQL400 (AS / 400 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Res-erved 9	Fld Length 10 11
12	X'70'	X'22'	X'10'	00000-00500(e)	1	0	8
12	X'70'	X'23'	X'90'	00000-00500(e)	1	0	8

QTDSQLX86 (Intel 80X86 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Res-erved 9	Fld Length 10 11
12	X'70'	X'22'	X'10'	00000-00850(e)	1	0	8
12	X'70'	X'23'	X'90'	00000-00850(e)	1	0	8

QTDSQLASC (IEEE ASCII Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Res-erved 9	Fld Length 10 11
12	X'70'	X'22'	X'10'	00000-00819(e)	1	0	8
12	X'70'	X'23'	X'90'	00000-00819(e)	1	0	8

QTDSQLVAX (VAX Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Res-erved 9	Fld Length 10 11
12	X'70'	X'22'	X'10'	00000-00819(e)	1	0	8
12	X'70'	X'23'	X'90'	00000-00819(e)	1	0	8

```
Example Descriptor in Hex     07780005   010123
(QTDSQL370 nullable form)     0C702390   000001F4  01000008
```

(e) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.

Figure 5-54 DRDA Type X'22,23' SQL Type 388,389 TIME

5.6.5.18 Timestamp

Length	Type	Identity	Class	MD Type	MD Ref Ty	DRDA Type
0	1	2	3	4	5	6

Meta Data (Environment-independent)

7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'24' (TS) X'25' (NTS)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID				Chr Siz	Res-erved	Fld	Length
0	1	2	3	4	5	6	7	8	9	10	11

QTDSQL370 (System / 370 Processors)

12	X'70'	X'24'	X'10'	00000-00500(e)	1	0	26
12	X'70'	X'25'	X'90'	00000-00500(e)	1	0	26

QTDSQL400 (AS / 400 Processors)

12	X'70'	X'24'	X'10'	00000-00500(e)	1	0	26
12	X'70'	X'25'	X'90'	00000-00500(e)	1	0	26

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'24'	X'10'	00000-00850(e)	1	0	26
12	X'70'	X'25'	X'90'	00000-00850(e)	1	0	26

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'24'	X'10'	00000-00819(e)	1	0	26
12	X'70'	X'25'	X'90'	00000-00819(e)	1	0	26

QTDSQLVAX (VAX Processors)

12	X'70'	X'24'	X'10'	00000-00819(e)	1	0	26
12	X'70'	X'25'	X'90'	00000-00819(e)	1	0	26

```
Example Descriptor in Hex      07780005   010125
(QTDSQL370 nullable form)      0C702590   000001F4  0100001A
```

(e) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.

Figure 5-55 DRDA Type X'24,25' SQL Type 392,393 TIMESTAMP

5.6.5.19 Fixed Bytes

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6

Meta Data (Environment-independent)

7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'26' (FB) X'27' (NFB)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11

QTDSQL370 (System / 370 Processors)

12 12	X'70' X'70'	X'26' X'27'	X'01' X'81'	0 0	0 0	0 0	32767 32767

QTDSQL400 (AS / 400 Processors)

12 12	X'70' X'70'	X'26' X'27'	X'01' X'81'	0 0	0 0	0 0	32767 32767

QTDSQLX86 (Intel 80X86 Processors)

12 12	X'70' X'70'	X'26' X'27'	X'01' X'81'	0 0	0 0	0 0	32767 32767

QTDSQLASC (IEEE ASCII Processors)

12 12	X'70' X'70'	X'26' X'27'	X'01' X'81'	0 0	0 0	0 0	32767 32767

QTDSQLVAX (VAX Processors)

12 12	X'70' X'70'	X'26' X'27'	X'01' X'81'	0 0	0 0	0 0	32767 32767

```
Example Descriptor in Hex      07780005   010127
(QTDSQL370 nullable form)      0C702781   00000000  00007FFF
```

Figure 5-56 DRDA Type X'26,27' SQL Type 452,453 FIXED BYTES

5.6.5.20 Variable Bytes

Meta Data (Environment-independent)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'28' (VB) X'29' (NVB)

QTDSQL370 (System / 370 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	Mode 9	Fld Length 10 11
12	X'70'	X'28'	X'02'	0	0	1	32767
12	X'70'	X'29'	X'82'	0	0	1	32767

QTDSQL400 (AS / 400 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	Mode 9	Fld Length 10 11
12	X'70'	X'28'	X'02'	0	0	1	32767
12	X'70'	X'29'	X'82'	0	0	1	32767

QTDSQLX86 (Intel 80X86 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	Mode 9	Fld Length 10 11
12	X'70'	X'28'	X'02'	0	0	1	32767
12	X'70'	X'29'	X'82'	0	0	1	32767

QTDSQLASC (IEEE ASCII Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	Mode 9	Fld Length 10 11
12	X'70'	X'28'	X'02'	0	0	1	32767
12	X'70'	X'29'	X'82'	0	0	1	32767

QTDSQLVAX (VAX Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	Mode 9	Fld Length 10 11
12	X'70'	X'28'	X'02'	0	0	1	32767
12	X'70'	X'29'	X'82'	0	0	1	32767

```
Example Descriptor in Hex      07780005   010129
(QTDSQL370 nullable form)      0C702982   00000000  00017FFF
```

Figure 5-57 DRDA Type X'28,29' SQL Type 448,449 VARIABLE BYTES

5.6.5.21 Long Variable Bytes

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6

Meta Data (Environment-independent)

7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'2A' (LVB) X'2B' (NLVB)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	Mode 8	Fld 9	Length 10 11

QTDSQL370 (System / 370 Processors)

12 12	X'70' X'70'	X'2A' X'2B'	X'02' X'82'	0 0	0 0	1 1	32767 32767

QTDSQL400 (AS / 400 Processors)

12 12	X'70' X'70'	X'2A' X'2B'	X'02' X'82'	0 0	0 0	1 1	32767 32767

QTDSQLX86 (Intel 80X86 Processors)

12 12	X'70' X'70'	X'2A' X'2B'	X'02' X'82'	0 0	0 0	1 1	32767 32767

QTDSQLASC (IEEE ASCII Processors)

12 12	X'70' X'70'	X'2A' X'2B'	X'02' X'82'	0 0	0 0	1 1	32767 32767

QTDSQLVAX (VAX Processors)

12 12	X'70' X'70'	X'2A' X'2B'	X'02' X'82'	0 0	0 0	1 1	32767 32767

```
Example Descriptor in Hex      07780005   01012B
(QTDSQL370 nullable form)      0C702B82   00000000  00017FFF
```

Figure 5-58 DRDA Type X'2A,2B' SQL Type 456,457 LONG VAR BYTES

5.6.5.22 Null-Terminated Bytes

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6

Meta Data (Environment-independent)

7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'2C' (NTB) X'2D' (NNTB)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	Mode 9	Fld Length 10 11

QTDSQL370 (System / 370 Processors)

12 12	X'70' X'70'	X'2C' X'2D'	X'03' X'83'	0 0	0 0	1 1	32767 32767

QTDSQL400 (AS / 400 Processors)

12 12	X'70' X'70'	X'2C' X'2D'	X'03' X'83'	0 0	0 0	1 1	32767 32767

QTDSQLX86 (Intel 80X86 Processors)

12 12	X'70' X'70'	X'2C' X'2D'	X'03' X'83'	0 0	0 0	1 1	32767 32767

QTDSQLASC (IEEE ASCII Processors)

12 12	X'70' X'70'	X'2C' X'2D'	X'03' X'83'	0 0	0 0	1 1	32767 32767

QTDSQLVAX (VAX Processors)

12 12	X'70' X'70'	X'2C' X'2D'	X'03' X'83'	0 0	0 0	1 1	32767 32767

```
Example Descriptor in Hex     07780005  01012D
(QTDSQL370 nullable form)     0C702D83  00000000  00017FFF
```

Figure 5-59 DRDA Type X'2C,2D' SQL Type 460,461 NULL-TERMINATED BYTES

5.6.5.23 Null-Terminated SBCS

Length	Type	Identity	Class	MD Type	MD Ref Ty	DRDA Type
0	1	2	3	4	5	6

Meta Data (Environment-independent)

7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'2E' (NTCS) X'2F' (NNTCS)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Siz	Mode	Fld Length
0	1	2	3	4 5 6 7	8	9	10 11

QTDSQL370 (System / 370 Processors)

12 12	X'70' X'70'	X'2E' X'2F'	X'14' X'94'	00000-00500(e) 00000-00500(e)	1 1	1 1	32767 32767

QTDSQL400 (AS / 400 Processors)

12 12	X'70' X'70'	X'2E' X'2F'	X'14' X'94'	00000-00500(e) 00000-00500(e)	1 1	1 1	32767 32767

QTDSQLX86 (Intel 80X86 Processors)

12 12	X'70' X'70'	X'2E' X'2F'	X'14' X'94'	00000-00850(e) 00000-00850(e)	1 1	1 1	32767 32767

QTDSQLASC (IEEE ASCII Processors)

12 12	X'70' X'70'	X'2E' X'2F'	X'14' X'94'	00000-00819(e) 00000-00819(e)	1 1	1 1	32767 32767

QTDSQLVAX (VAX Processors)

12 12	X'70' X'70'	X'2E' X'2F'	X'14' X'94'	00000-00819(e) 00000-00819(e)	1 1	1 1	32767 32767

```
Example Descriptor in Hex      07780005   01012F
(QTDSQL370 nullable form)      0C702F94   000001F4  01017FFF
```

(e) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.

Figure 5-60 DRDA Type X'2E,2F' SQL Type 460,461 NULL-TERMINATED SBCS

5.6.5.24 Fixed Character SBCS

Length	Type	Identity	Class	MD Type	MD Ref Ty	DRDA Type
0	1	2	3	4	5	6

Meta Data (Environment-independent)

7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'30' (FCS) X'31' (NFCS)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Siz	Res-erved	Fld Length
0	1	2	3	4 5 6 7	8	9	10 11

QTDSQL370 (System / 370 Processors)

12	X'70'	X'30'	X'10'	00000-00500(e)	1	0	32767
12	X'70'	X'31'	X'90'	00000-00500(e)	1	0	32767

QTDSQL400 (AS / 400 Processors)

12	X'70'	X'30'	X'10'	00000-00500(e)	1	0	32767
12	X'70'	X'31'	X'90'	00000-00500(e)	1	0	32767

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'30'	X'10'	00000-00850(e)	1	0	32767
12	X'70'	X'31'	X'90'	00000-00850(e)	1	0	32767

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'30'	X'10'	00000-00819(e)	1	0	32767
12	X'70'	X'31'	X'90'	00000-00819(e)	1	0	32767

QTDSQLVAX (VAX Processors)

12	X'70'	X'30'	X'10'	00000-00819(e)	1	0	32767
12	X'70'	X'31'	X'90'	00000-00819(e)	1	0	32767

```
Example Descriptor in Hex     07780005  010131
(QTDSQL370 nullable form)     0C703190  000001F4  01007FFF
```

(e) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.

Figure 5-61 DRDA Type X'30,31' SQL Type 452,453 FIXED CHARACTER SBCS

5.6.5.25 Variable Character SBCS

Meta Data (Environment-independent)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'32' (VCS) X'33' (NVCS)

QTDSQL370 (System / 370 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Mode 9	Fld Length 10 11
12	X'70'	X'32'	X'11'	00000-00500(e)	1	1	32767
12	X'70'	X'33'	X'91'	00000-00500(e)	1	1	32767

QTDSQL400 (AS / 400 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Mode 9	Fld Length 10 11
12	X'70'	X'32'	X'11'	00000-00500(e)	1	1	32767
12	X'70'	X'33'	X'91'	00000-00500(e)	1	1	32767

QTDSQLX86 (Intel 80X86 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Mode 9	Fld Length 10 11
12	X'70'	X'32'	X'11'	00000-00850(e)	1	1	32767
12	X'70'	X'33'	X'91'	00000-00850(e)	1	1	32767

QTDSQLASC (IEEE ASCII Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Mode 9	Fld Length 10 11
12	X'70'	X'32'	X'11'	00000-00819(e)	1	1	32767
12	X'70'	X'33'	X'91'	00000-00819(e)	1	1	32767

QTDSQLVAX (VAX Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Mode 9	Fld Length 10 11
12	X'70'	X'32'	X'11'	00000-00819(e)	1	1	32767
12	X'70'	X'33'	X'91'	00000-00819(e)	1	1	32767

```
Example Descriptor in Hex      07780005   010133
(QTDSQL370 nullable form)      0C703391   000001F4  01017FFF
```

(e) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.

Figure 5-62 DRDA Type X'32,33' SQL Type 448,449 VARIABLE CHARACTER SBCS

5.6.5.26 Long Variable Character SBCS

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
Meta Data (Environment-independent)						
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'34' (LVCS) X'35' (NLVCS)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Mode 9	Fld Length 10 11
QTDSQL370 (System / 370 Processors)							
12	X'70'	X'34'	X'11'	00000-00500(e)	1	1	32767
12	X'70'	X'35'	X'91'	00000-00500(e)	1	1	32767
QTDSQL400 (AS / 400 Processors)							
12	X'70'	X'34'	X'11'	00000-00500(e)	1	1	32767
12	X'70'	X'35'	X'91'	00000-00500(e)	1	1	32767
QTDSQLX86 (Intel 80X86 Processors)							
12	X'70'	X'34'	X'11'	00000-00850(e)	1	1	32767
12	X'70'	X'35'	X'91'	00000-00850(e)	1	1	32767
QTDSQLASC (IEEE ASCII Processors)							
12	X'70'	X'34'	X'11'	00000-00819(e)	1	1	32767
12	X'70'	X'35'	X'91'	00000-00819(e)	1	1	32767
QTDSQLVAX (VAX Processors)							
12	X'70'	X'34'	X'11'	00000-00819(e)	1	1	32767
12	X'70'	X'35'	X'91'	00000-00819(e)	1	1	32767

```
Example Descriptor in Hex     07780005  010135
(QTDSQL370 nullable form)     0C703591  000001F4  01017FFF
```

(e) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.

Figure 5-63 DRDA Type X'34,35' SQL Type 456,457 LONG VAR CHARACTER SBCS

5.6.5.27 Fixed-Character DBCS (GRAPHIC)

Length	Type	Identity	Class	MD Type	MD Ref Ty	DRDA Type
0	1	2	3	4	5	6

Meta Data (Environment-independent)

7	MDD X’78’	0	Rel.DB X’05’	Data Type X’01’	Next Byte X’01’	X’36’ (FCD) X’37’ (NFCD)

FD Tr Ln	FD:OCA Tripl ’SDA’	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Siz	Res-erved	Fld Length
0	1	2	3	4 5 6 7	8	9	10 11

QTDSQL370 (System / 370 Processors)

12 12	X’70’ X’70’	X’36’ X’37’	X’10’ X’90’	00000-00300(f) 00000-00300(f)	2 2	0 0	16383 16383

QTDSQL400 (AS / 400 Processors)

12 12	X’70’ X’70’	X’36’ X’37’	X’10’ X’90’	00000-00300(f) 00000-00300(f)	2 2	0 0	16383 16383

QTDSQLX86 (Intel 80X86 Processors)

12 12	X’70’ X’70’	X’36’ X’37’	X’10’ X’90’	00000-00301(f) 00000-00301(f)	2 2	0 0	16383 16383

QTDSQLASC (IEEE ASCII Processors)

12 12	X’70’ X’70’	X’36’ X’37’	X’10’ X’90’	00000-01200(f) 00000-01200(f)	2 2	0 0	16383 16383

QTDSQLVAX (VAX Processors)

12 12	X’70’ X’70’	X’36’ X’37’	X’10’ X’90’	00000-01200(f) 00000-01200(f)	2 2	0 0	16383 16383

```
Example Descriptor in Hex      07780005   010137
(QTDSQL370 nullable form)      0C703790   0000012C  02003FFF
```

(f) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.

Figure 5-64 DRDA Type X‘36,37’ SQL Type 468,469 FIXED CHARACTER DBCS

5.6.5.28 Variable-Character DBCS (GRAPHIC)

Meta Data (Environment-independent)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'38' (VCD) X'39' (NVCD)

QTDSQL370 (System / 370 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Mode 9	Fld Length 10 11
12	X'70'	X'38'	X'11'	00000-00300(f)	2	1	16383
12	X'70'	X'39'	X'91'	00000-00300(f)	2	1	16383

QTDSQL400 (AS / 400 Processors)

12	X'70'	X'38'	X'11'	00000-00300(f)	2	1	16383
12	X'70'	X'39'	X'91'	00000-00300(f)	2	1	16383

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'38'	X'11'	00000-00301(f)	2	1	16383
12	X'70'	X'39'	X'91'	00000-00301(f)	2	1	16383

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'38'	X'11'	00000-01200(f)	2	1	16383
12	X'70'	X'39'	X'91'	00000-01200(f)	2	1	16383

QTDSQLVAX (VAX Processors)

12	X'70'	X'38'	X'11'	00000-01200(f)	2	1	16383
12	X'70'	X'39'	X'91'	00000-01200(f)	2	1	16383

```
Example Descriptor in Hex      07780005   010139
(QTDSQL370 nullable form)      0C703991   0000012C  02013FFF
```

(f) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.

Figure 5-65 DRDA Type X'38,39' SQL Type 464,465 VARIABLE CHARACTER DBCS

5.6.5.29 Long Variable Character DBCS (GRAPHIC)

Meta Data (Environment-independent)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'3A' (LVCD) X'3B' (NLVCD)

QTDSQL370 (System / 370 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Mode 9	Fld Length 10 11
12	X'70'	X'3A'	X'11'	00000-00300(f)	2	1	16383
12	X'70'	X'3B'	X'91'	00000-00300(f)	2	1	16383

QTDSQL400 (AS / 400 Processors)

12	X'70'	X'3A'	X'11'	00000-00300(f)	2	1	16383
12	X'70'	X'3B'	X'91'	00000-00300(f)	2	1	16383

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'3A'	X'11'	00000-00301(f)	2	1	16383
12	X'70'	X'3B'	X'91'	00000-00301(f)	2	1	16383

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'3A'	X'11'	00000-01200(f)	2	1	16383
12	X'70'	X'3B'	X'91'	00000-01200(f)	2	1	16383

QTDSQLVAX (VAX Processors)

12	X'70'	X'3A'	X'11'	00000-01200(f)	2	1	16383
12	X'70'	X'3B'	X'91'	00000-01200(f)	2	1	16383

```
Example Descriptor in Hex      07780005   01013B
(QTDSQL370 nullable form)      0C703B91   0000012C   02013FFF
```

(f) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.

Figure 5-66 DRDA Type X'3A,3B' SQL Type 472,473 LONG VAR CHARACTER DBCS

5.6.5.30 Fixed Character Mixed

Length	Type	Identity	Class	MD Type	MD Ref Ty	DRDA Type
0	1	2	3	4	5	6

Meta Data (Environment-independent)

7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'3C' (FCM) X'3D' (NFCM)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Siz	Res-erved	Fld Length
0	1	2	3	4 5 6 7	8	9	10 11

QTDSQL370 (System / 370 Processors)

12	X'70'	X'3C'	X'10'	00000-00930(g)	1	0	32767
12	X'70'	X'3D'	X'90'	00000-00930(g)	1	0	32767

QTDSQL400 (AS / 400 Processors)

12	X'70'	X'3C'	X'10'	00000-00930(g)	1	0	32767
12	X'70'	X'3D'	X'90'	00000-00930(g)	1	0	32767

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'3C'	X'10'	00000-00932(g)	1	0	32767
12	X'70'	X'3D'	X'90'	00000-00932(g)	1	0	32767

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'3C'	X'10'	00000-01200(g)	1	0	32767
12	X'70'	X'3D'	X'90'	00000-01200(g)	1	0	32767

QTDSQLVAX (VAX Processors)

12	X'70'	X'3C'	X'10'	00000-01200(g)	1	0	32767
12	X'70'	X'3D'	X'90'	00000-01200(g)	1	0	32767

```
Example Descriptor in Hex       07780005   01013D
(QTDSQL370 nullable form)       0C703D90  000003A2  01007FFF
```

(g) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.

Figure 5-67 DRDA Type X'3C,3D' SQL Type 452,453 FIXED CHARACTER MIXED

5.6.5.31 Variable Character Mixed

Meta Data (Environment-independent)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'3E' (VCM) X'3F' (NVCM)

QTDSQL370 (System / 370 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Mode 9	Fld Length 10 11
12	X'70'	X'3E'	X'11'	00000-00930(g)	1	1	32767
12	X'70'	X'3F'	X'91'	00000-00930(g)	1	1	32767

QTDSQL400 (AS / 400 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Mode 9	Fld Length 10 11
12	X'70'	X'3E'	X'11'	00000-00930(g)	1	1	32767
12	X'70'	X'3F'	X'91'	00000-00930(g)	1	1	32767

QTDSQLX86 (Intel 80X86 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Mode 9	Fld Length 10 11
12	X'70'	X'3E'	X'11'	00000-00932(g)	1	1	32767
12	X'70'	X'3F'	X'91'	00000-00932(g)	1	1	32767

QTDSQLASC (IEEE ASCII Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Mode 9	Fld Length 10 11
12	X'70'	X'3E'	X'11'	00000-01200(g)	1	1	32767
12	X'70'	X'3F'	X'91'	00000-01200(g)	1	1	32767

QTDSQLVAX (VAX Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Mode 9	Fld Length 10 11
12	X'70'	X'3E'	X'11'	00000-01200(g)	1	1	32767
12	X'70'	X'3F'	X'91'	00000-01200(g)	1	1	32767

Example Descriptor in Hex	07780005 01013F
(QTDSQL370 nullable form)	0C703F91 000003A2 01017FFF

(g) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.

Figure 5-68 DRDA Type X'3E,3F' SQL Type 448,449 VARIABLE CHARACTER MIXED

5.6.5.32 Long Variable Character Mixed

Meta Data (Environment-independent)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'40' (LVCM) X'41' (NLVCM)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Mode 9	Fld Length 10 11
QTDSQL370 (System / 370 Processors)							
12	X'70'	X'40'	X'11'	00000-00930(g)	1	1	32767
12	X'70'	X'41'	X'91'	00000-00930(g)	1	1	32767
QTDSQL400 (AS / 400 Processors)							
12	X'70'	X'40'	X'11'	00000-00930(g)	1	1	32767
12	X'70'	X'41'	X'91'	00000-00930(g)	1	1	32767
QTDSQLX86 (Intel 80X86 Processors)							
12	X'70'	X'40'	X'11'	00000-00932(g)	1	1	32767
12	X'70'	X'41'	X'91'	00000-00932(g)	1	1	32767
QTDSQLASC (IEEE ASCII Processors)							
12	X'70'	X'40'	X'11'	00000-01200(g)	1	1	32767
12	X'70'	X'41'	X'91'	00000-01200(g)	1	1	32767
QTDSQLVAX (VAX Processors)							
12	X'70'	X'40'	X'11'	00000-01200(g)	1	1	32767
12	X'70'	X'41'	X'91'	00000-01200(g)	1	1	32767

```
Example Descriptor in Hex       07780005   010141
(QTDSQL370 nullable form)       0C704191   000003A2  01017FFF
```

(g) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.

Figure 5-69 DRDA Type X'40,41' SQL Type 456,457 LONG VARIABLE CHARACTER MIXED

5.6.5.33 Null-Terminated Mixed

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6

Meta Data (Environment-independent)

7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'42' (NTM) X'43' (NNTM)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Mode 9	Fld Length 10 11

QTDSQL370 (System / 370 Processors)

12 12	X'70' X'70'	X'42' X'43'	X'14' X'94'	00000-00930(g) 00000-00930(g)	1 1	1 1	32767 32767

QTDSQL400 (AS / 400 Processors)

12 12	X'70' X'70'	X'42' X'43'	X'14' X'94'	00000-00930(g) 00000-00930(g)	1 1	1 1	32767 32767

QTDSQLX86 (Intel 80X86 Processors)

12 12	X'70' X'70'	X'42' X'43'	X'14' X'94'	00000-00932(g) 00000-00932(g)	1 1	1 1	32767 32767

QTDSQLASC (IEEE ASCII Processors)

12 12	X'70' X'70'	X'42' X'43'	X'14' X'94'	00000-01200(g) 00000-01200(g)	1 1	1 1	32767 32767

QTDSQLVAX (VAX Processors)

12 12	X'70' X'70'	X'42' X'43'	X'14' X'94'	00000-01200(g) 00000-01200(g)	1 1	1 1	32767 32767

```
Example Descriptor in Hex      07780005   010143
(QTDSQL370 nullable form)      0C704394   000003A2  01017FFF
```

(g) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.

Figure 5-70 DRDA Type X'42,43' SQL Type 460,461 NULL-TERMINATED MIXED

5.6.5.34 Pascal L String Bytes

Length	Type	Identity	Class	MD Type	MD Ref Ty	DRDA Type
0	1	2	3	4	5	6

Meta Data (Environment-independent)

7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'44' (PLB) X'45' (NPLB)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	Reserved					Mode	Fld	Length
0	1	2	3	4	5	6	7	8	9	10	11

QTDSQL370 (System / 370 Processors)

12 12	X'70' X'70'	X'44' X'45'	X'07' X'87'	0 0	0 0	1 1	255 255

QTDSQL400 (AS / 400 Processors)

12 12	X'70' X'70'	X'44' X'45'	X'07' X'87'	0 0	0 0	1 1	255 255

QTDSQLX86 (Intel 80X86 Processors)

12 12	X'70' X'70'	X'44' X'45'	X'07' X'87'	0 0	0 0	1 1	255 255

QTDSQLASC (IEEE ASCII Processors)

12 12	X'70' X'70'	X'44' X'45'	X'07' X'87'	0 0	0 0	1 1	255 255

QTDSQLVAX (VAX Processors)

12 12	X'70' X'70'	X'44' X'45'	X'07' X'87'	0 0	0 0	1 1	255 255

```
Example Descriptor in Hex      07780005   010145
(QTDSQL370 nullable form)      0C704587   00000000  000100FF
```

(e) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.

Figure 5-71 DRDA Type X'44,45' SQL Type 476,477 PASCAL L STRING BYTES

5.6.5.35 Pascal L String SBCS

Length	Type	Identity	Class	MD Type	MD Ref Ty	DRDA Type
0	1	2	3	4	5	6

Meta Data (Environment-independent)

7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'46' (PLS) X'47' (NPLS)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Siz	Mode	Fld Length
0	1	2	3	4 5 6 7	8	9	10 11

QTDSQL370 (System / 370 Processors)

12	X'70'	X'46'	X'19'	00000-00500(e)	1	1	255
12	X'70'	X'47'	X'99'	00000-00500(e)	1	1	255

QTDSQL400 (AS / 400 Processors)

12	X'70'	X'46'	X'19'	00000-00500(e)	1	1	255
12	X'70'	X'47'	X'99'	00000-00500(e)	1	1	255

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'46'	X'19'	00000-00850(e)	1	1	255
12	X'70'	X'47'	X'99'	00000-00850(e)	1	1	255

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'46'	X'19'	00000-00819(e)	1	1	255
12	X'70'	X'47'	X'99'	00000-00819(e)	1	1	255

QTDSQLVAX (VAX Processors)

12	X'70'	X'46'	X'19'	00000-00819(e)	1	1	255
12	X'70'	X'47'	X'99'	00000-00819(e)	1	1	255

```
Example Descriptor in Hex     07780005   010147
(QTDSQL370 nullable form)     0C704799   000001F4  010100FF
```

(e) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.

Figure 5-72 DRDA Type X'46,47' SQL Type 476,477 PASCAL L STRING SBCS

5.6.5.36 Pascal L String Mixed

Meta Data (Environment-independent)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X’78’	0	Rel.DB X’05’	Data Type X’01’	Next Byte X’01’	X’48’ (PLM) X’49’ (NPLM)

QTDSQL370 (System / 370 Processors)

FD Tr Ln 0	FD:OCA Tripl ’SDA’ 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Mode 9	Fld Length 10 11
12	X’70’	X’48’	X’19’	00000-00930(g)	1	1	255
12	X’70’	X’49’	X’99’	00000-00930(g)	1	1	255

QTDSQL400 (AS / 400 Processors)

12	X’70’	X’48’	X’19’	00000-00930(g)	1	1	255
12	X’70’	X’49’	X’99’	00000-00930(g)	1	1	255

QTDSQLX86 (Intel 80X86 Processors)

12	X’70’	X’48’	X’19’	00000-00932(g)	1	1	255
12	X’70’	X’49’	X’99’	00000-00932(g)	1	1	255

QTDSQLASC (IEEE ASCII Processors)

12	X’70’	X’48’	X’19’	00000-01200(g)	1	1	255
12	X’70’	X’49’	X’99’	00000-01200(g)	1	1	255

QTDSQLVAX (VAX Processors)

12	X’70’	X’48’	X’19’	00000-01200(g)	1	1	255
12	X’70’	X’49’	X’99’	00000-01200(g)	1	1	255

```
Example Descriptor in Hex      07780005   010149
(QTDSQL370 nullable form)      0C704999   000003A2  010100FF
```

(g) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.

Figure 5-73 DRDA Type X‘48,49’ SQL Type 476,477 PASCAL L STRING MIXED

5.6.5.37 SBCS Datalink

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6

Meta Data (Environment-independent)

7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'4C' (DLS) X'4D' (NDLS)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Mode 9	Fld Length 10 11	

QTDSQL370 (System / 370 Processors)

12	X'70'	X'4C'	X'11'	00000-00500(e)	1	1	32767	(x)
12	X'70'	X'4D'	X'91'	00000-00500(e)	1	1	32767	(x)

QTDSQL400 (AS / 400 Processors)

12	X'70'	X'4C'	X'11'	00000-00500(e)	1	1	32767	(x)
12	X'70'	X'4D'	X'91'	00000-00500(e)	1	1	32767	(x)

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'4C'	X'11'	00000-00850(e)	1	1	32767	(x)
12	X'70'	X'4D'	X'91'	00000-00850(e)	1	1	32767	(x)

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'4C'	X'11'	00000-00819(e)	1	1	32767	(x)
12	X'70'	X'4D'	X'91'	00000-00819(e)	1	1	32767	(x)

QTDSQLVAX (VAX Processors)

12	X'70'	X'4C'	X'11'	00000-00819(e)	1	1	32767	(x)
12	X'70'	X'4D'	X'91'	00000-00819(e)	1	1	32767	(x)

```
Example Descriptor in Hex      07780005  01014D
(QTDSQL370 nullable form)      0C704D91  000001F4  01017FFF
```

(e) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.
(x) The contents of this VARCHAR-like data type
must conform to DRDA rule DT20.

Figure 5-74 DRDA Type X'4C,4D' SQL Type 396,397 SBCS DATALINK

5.6.5.38 Mixed-Byte Datalink

Length	Type	Identity	Class	MD Type	MD Ref Ty	DRDA Type
0	1	2	3	4	5	6

Meta Data (Environment-independent)

7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'4E' (DLM) X'4F' (NDLM)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Siz	Mode	Fld Length	
0	1	2	3	4 5 6 7	8	9	10 11	

QTDSQL370 (System / 370 Processors)

12	X'70'	X'4E'	X'11'	00000-00930(e)	1	1	32767	(x)
12	X'70'	X'4F'	X'91'	00000-00930(e)	1	1	32767	(x)

QTDSQL400 (AS / 400 Processors)

12	X'70'	X'4E'	X'11'	00000-00930(e)	1	1	32767	(x)
12	X'70'	X'4F'	X'91'	00000-00930(e)	1	1	32767	(x)

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'4E'	X'11'	00000-00932(e)	1	1	32767	(x)
12	X'70'	X'4F'	X'91'	00000-00932(e)	1	1	32767	(x)

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'4E'	X'11'	00000-01200(e)	1	1	32767	(x)
12	X'70'	X'4F'	X'91'	00000-01200(e)	1	1	32767	(x)

QTDSQLVAX (VAX Processors)

12	X'70'	X'4E'	X'11'	00000-01200(e)	1	1	32767	(x)
12	X'70'	X'4F'	X'91'	00000-01200(e)	1	1	32767	(x)

```
Example Descriptor in Hex      07780005   01014F
(QTDSQL370 nullable form)      0C704F91   00000930  01017FFF
```

(e) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.
(x) The contents of this VARCHAR-like data type
must conform to DRDA rule DT20.

Figure 5-75 DRDA Type X'4E,4F' SQL Type 396,397 MIXED-BYTE DATALINK

5.6.5.39 Large Object Bytes

Meta Data (Environment-independent)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'C8' (OB) X'C9' (NOB)

QTDSQL370 (System / 370 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X'70'	X'C8'	X'50'	0	0	1	8 (h)
12	X'70'	X'C9'	X'D0'	0	0	1	8 (h)

QTDSQL400 (AS / 400 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X'70'	X'C8'	X'50'	0	0	1	8 (h)
12	X'70'	X'C9'	X'D0'	0	0	1	8 (h)

QTDSQLX86 (Intel 80X86 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X'70'	X'C8'	X'50'	0	0	1	8 (h)
12	X'70'	X'C9'	X'D0'	0	0	1	8 (h)

QTDSQLASC (IEEE ASCII Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X'70'	X'C8'	X'50'	0	0	1	8 (h)
12	X'70'	X'C9'	X'D0'	0	0	1	8 (h)

QTDSQLVAX (VAX Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Reserved 4 5 6 7	8	9	Fld Length 10 11
12	X'70'	X'C8'	X'50'	0	0	0	8 (h)
12	X'70'	X'C9'	X'D0'	0	0	0	8 (h)

```
Example Descriptor in Hex      07780005   0101C9
(QTDSQL370 nullable form)      0C70C9D0 00000000  00018008
```

(h) The placeholder indicator bit is set to '1'B.

Figure 5-76 DRDA Type X'C8,C9' SQL Type 404,405 LARGE OBJECT BYTES

5.6.5.40 Large Object Character SBCS

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6

Meta Data (Environment-independent)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'CA' (OCS) X'CB' (NOCS)

QTDSQL370 (System / 370 Processors)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Mode 9	Fld Length 10 11
12 12	X'70' X'70'	X'CA' X'CB'	X'51' X'D1'	00000-00500(e) 00000-00500(e)	1 1	1 1	8 (h) 8 (h)

QTDSQL400 (AS / 400 Processors)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Siz	Mode	Fld Length
12 12	X'70' X'70'	X'CA' X'CB'	X'51' X'D1'	00000-00500(e) 00000-00500(e)	1 1	1 1	8 (h) 8 (h)

QTDSQLX86 (Intel 80X86 Processors)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Siz	Mode	Fld Length
12 12	X'70' X'70'	X'CA' X'CB'	X'51' X'D1'	00000-00850(e) 00000-00850(e)	1 1	1 1	8 (h) 8 (h)

QTDSQLASC (IEEE ASCII Processors)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Siz	Mode	Fld Length
12 12	X'70' X'70'	X'CA' X'CB'	X'51' X'D1'	00000-00819(e) 00000-00819(e)	1 1	1 1	8 (h) 8 (h)

QTDSQLVAX (VAX Processors)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Siz	Mode	Fld Length
12 12	X'70' X'70'	X'CA' X'CB'	X'51' X'D1'	00000-00819(e) 00000-00819(e)	1 1	1 1	8 (h) 8 (h)

```
Example Descriptor in Hex      07780005   0101CB
(QTDSQL370 nullable form)      0C70CBD1000001F4  01018008
```

(e) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.
(h) The placeholder indicator bit is set to '1'B.

Figure 5-77 DRDA Type X'CA,CB' SQL Type 408,409 LARGE OBJECT CHAR. SBCS

5.6.5.41 Large Object Character DBCS (GRAPHIC)

Meta Data (Environment-independent)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'CC' (OCD) X'CD' (NOCD)

FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	CCSID 4 5 6 7	Chr Siz 8	Mode 9	Fld Length 10 11
QTDSQL370 (System / 370 Processors)							
12 12	X'70' X'70'	X'CC' X'CD'	X'51' X'D1'	00000-00300(f) 00000-00300(f)	2 2	1 1	8 (h) 8 (h)
QTDSQL400 (AS / 400 Processors)							
12 12	X'70' X'70'	X'CC' X'CD'	X'51' X'D1'	00000-00300(f) 00000-00300(f)	2 2	1 1	8 (h) 8 (h)
QTDSQLX86 (Intel 80X86 Processors)							
12 12	X'70' X'70'	X'CC' X'CD'	X'51' X'D1'	00000-00301(f) 00000-00301(f)	2 2	1 1	8 (h) 8 (h)
QTDSQLASC (IEEE ASCII Processors)							
12 12	X'70' X'70'	X'CC' X'CD'	X'51' X'D1'	00000-01200(f) 00000-01200(f)	2 2	1 1	8 (h) 8 (h)
QTDSQLVAX (VAX Processors)							
12 12	X'70' X'70'	X'CC' X'CD'	X'51' X'D1'	00000-01200(f) 00000-01200(f)	2 2	1 1	8 (h) 8 (h)

```
Example Descriptor in Hex      07780005    0101CD
(QTDSQL370 nullable form)      0C70CDD1    0000012C  02018008
```

(f) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.
(h) The placeholder indicator bit is set to '1'B.

Figure 5-78 DRDA Type X'CC,CD' SQL Type 412,413 LARGE OBJECT CHAR. DBCS

5.6.5.42 Large Object Character Mixed

Length	Type	Identity	Class	MD Type	MD Ref Ty	DRDA Type
0	1	2	3	4	5	6

Meta Data (Environment-independent)

7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	X'CE' (OCM) X'CF' (NOCM)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID				Chr Siz	Mode	Fld	Length
0	1	2	3	4	5	6	7	8	9	10	11

QTDSQL370 (System / 370 Processors)

12	X'70'	X'CE'	X'51'	00000-00930(g)	1	1	8 (h)
12	X'70'	X'CF'	X'D1'	00000-00930(g)	1	1	8 (h)

QTDSQL400 (AS / 400 Processors)

12	X'70'	X'CE'	X'51'	00000-00930(g)	1	1	8 (h)
12	X'70'	X'CF'	X'D1'	00000-00930(g)	1	1	8 (h)

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'CE'	X'51'	00000-00932(g)	1	1	8 (h)
12	X'70'	X'CF'	X'D1'	00000-00932(g)	1	1	8 (h)

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'CE'	X'51'	00000-01200(g)	1	1	8 (h)
12	X'70'	X'CF'	X'D1'	00000-01200(g)	1	1	8 (h)

QTDSQLVAX (VAX Processors)

12	X'70'	X'CE'	X'51'	00000-01200(g)	1	1	8 (h)
12	X'70'	X'CF'	X'D1'	00000-01200(g)	1	1	8 (h)

```
Example Descriptor in Hex      07780005    0101CF
(QTDSQL370 nullable form)      0C70CFD1    000003A2  01018008
```

(g) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.
(h) The placeholder indicator bit is set to '1'B.

Figure 5-79 DRDA Type X'CE,CF' SQL Type 408,409 LARGE OBJECT CHAR. MIXED

5.6.6 Late Environmental Descriptors

DRDA does not define environmental descriptors that are used exclusively as Late Environmental Descriptors. These descriptors are provided late because of a specific representational situation that could not be determined until the user's data was examined.

The Late Environmental Descriptors are constructed from an MDD triplet (to specify the required DRDA semantics) and an SDA to describe the representation desired. In every case, the MDD entry is exactly like the one for the DRDA type being overridden. An appropriately different SDA follows this MDD.

Consider the following situation. An application running in the OS/2 environment is using the extended box drawing characters provided in Character Set 919 in Code Page 437 (CCSID 437 defines this). The rest of the operations of the database manager are in Multilingual Latin-1 characters (CCSID 850). CCSID 850 would be specified in the TYPDEFOVR parameter that flows with the ACCRDB DDM command at the time that a connection is made to the appropriate server.

The fields containing the boxes are fixed-length character fields containing data coded in a Single-Byte Character Set. Figure 5-80 is the standard representation for this information in this environment. (This is taken from Section 5.6.5.24 on page 224.)

Length	Type	Identity	Class	MD Type	MD Ref Ty	DRDA Type
0	1	2	3	4	5	6

Meta Data (Environment-independent)

7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'01'	DRDA Type X'30' (FCS)

FD	FD:OCA	FD:OCA	FD:OCA				Prec;/Scale	
Tr	Tripl	Tripl	Field	Res-		Chr	or	
Ln	'SDA'	LID	Type	erved CCSID		Siz	Mode	Fld Length
0	1	2	3	4 5 6 7	8	9	10 11	

QTDSQLX86 (Intel 80X86 Processors)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	Reserved / CCSID	Chr Siz	Mode	Prec;/Scale or Fld Length
0	1	2	3	4 5 6 7	8	9	10 11
12	X'70'	X'30'	X'10'	00000-00850(e)	1	0	32767

```
Example Descriptor in Hex       07780005  010130
(QTDSQLX86 nullable form)       0C703010  00000352  01007FFF
```

(e) The CCSID specified here is an example.
The actual CCSID is specified via a DDM
TYPDEFOVR parameter or object,
or by a late environmental descriptor.

Figure 5-80 DRDA Type X'30', SQL Type 468, MDD Override Example—Base

The definition in Figure 5-81 on page 244 specifies the other character set needed to properly represent the box drawing character data.

Meta Data (Environment-independent)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Ty 5	DRDA Type 6
7	MDD X’78’	0	Rel.DB X’05’	Data Type X’01’	Next Byte X’01’	DRDA Type X’30’ (FCS)

QTDSQLX86 (Intel 80X86 Processors)

FD Tr Ln 0	FD:OCA Tripl ’SDA’ 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Res-erved 4 5	CCSID 6 7	Chr Siz 8	Mode 9	Prec;/Scale or Fld Length 10 11
12	X’70’	X’99’	X’10’	00000-00437(h)		1	0	32767

Example Descriptor in Hex	07780005 010130
(QTDSQLX86 nullable form)	0C709910 000001B5 01007FFF

(h) As a late descriptor, this CCSID value overrides the TYPDEFOVR that flows with ACCRDB.

Figure 5-81 DRDA Type X‘30’, SQL Type 468, MDD Override Example—Override

Only the SDA part of the descriptor has changed. In the original descriptor, LID X’30’ specified 850 as the CCSID. In the new descriptor, LID X’99’ specifies 437 as the CCSID.

The MDD specification is exactly the same for both. They are both DRDA Fixed-Length Single Byte Character Set strings.

When the application requester or application server assembles the user data group descriptor, references to LID X‘30’ imply SBCS data encoded in the standard way. References to LID X’99’ imply SBCS data encoded using the specially defined CCSID. Both types of data can be included in the same row of user data. As many occurrences of either type as are necessary to describe the data are included in the GDA triplet that defines the group.

Section 5.7.1 on page 250 provides more discussion of overriding descriptors.

This concludes the detailed discussion of building DRDA descriptor triplets. The remainder of this chapter lists descriptors and examples in the order that the triplets must be assembled to be processed correctly. That is, Environmental Descriptors precede Group Descriptors, which precede Row Descriptors, which precede Array Descriptors. Early descriptors precede late descriptors.

5.7 FD:OCA Meta Data Summary

A data unit is the link representation of something that can be in a control block in storage. DRDA defines the data units. SQL or the implementing product defines the control blocks.

DRDA uses the FD:OCA Meta Data Definition (MDD) to relate DRDA types and data units to their FD:OCA representations. FD:OCA has defined the value 5 as the application class for relational database. DRDA defines the meta data types and meta data references within that application class.

DRDA defines four meta data types. These types are:

1. Relate DRDA and SQL data types to their representations.
2. Relate names of group data units to their representations.
3. Relate names of single row data units to their representations.
4. Relate names of array data units to their representations.

DRDA reserves all other meta data type values within application class 5 for future use.

Within each meta data type, DRDA provides a coded value as the meta data reference. Each of these values corresponds to a particular data type or data unit. All meta data reference values not shown in the tables below are reserved.

The following tables show all valid values that DRDA defines.

Table 5-12 MDD References Used in Early Environmental Descriptors

Application Class	Meta Data Type	Meta Data Reference DRDA-Type and Name	SQL Data Type	Description
X‘05’	X‘01’	X‘02’(I4)	496	4-byte Integer
X‘05’	X‘01’	X‘03’(NI4)	497	Nullable 4-byte Integer
X‘05’	X‘01’	X‘04’(I2)	500	2-byte Integer
X‘05’	X‘01’	X‘05’(NI2)	501	Nullable 2-byte Integer
X‘05’	X‘01’	X‘06’(I1)	n/a	1-byte Integer
X‘05’	X‘01’	X‘07’(NI1)	n/a	Nullable 1-byte Integer
X‘05’	X‘01’	X‘08’(BF16)	(480)	16-byte Binary Floating Point
X‘05’	X‘01’	X‘09’(NBF16)	(481)	Nullable 16-byte Binary Floating Point
X‘05’	X‘01’	X‘0A’(BF8)	480	8-byte Binary Floating Point
X‘05’	X‘01’	X‘0B’(NBF8)	481	Nullable 8-byte Binary Floating Point
X‘05’	X‘01’	X‘0C’(BF4)	480	4-byte Binary Floating Point
X‘05’	X‘01’	X‘0D’(NBF4)	481	Nullable 4-byte Binary Floating Point
X‘05’	X‘01’	X‘0E’(FD)	484	Fixed Decimal
X‘05’	X‘01’	X‘0F’(NFD)	485	Nullable Fixed Decimal
X‘05’	X‘01’	X‘10’(ZD)	488	Zoned Decimal
X‘05’	X‘01’	X‘11’(NZD)	489	Nullable Zoned Decimal
X‘05’	X‘01’	X‘12’(N)	504	Numeric Character
X‘05’	X‘01’	X‘13’(NN)	505	Nullable Numeric Character
X‘05’	X‘01’	X‘14’(RSL)	972	Result Set Locator

Application Class	Meta Data Type	Meta Data Reference DRDA-Type and Name	SQL Data Type	Description
X‘05’	X‘01’	X‘15’(NRSL)	973	Nullable Result Set Locator
X‘05’	X‘01’	X‘16’(I8)	492	Eight-byte Integer
X‘05’	X‘01’	X‘17’(NI8)	493	Nullable Eight-byte Integer
X‘05’	X‘01’	X‘18’(OBL)	960	Large Object Bytes Locator
X‘05’	X‘01’	X‘19’(NOBL)	961	Nullable Large Object Bytes Locator
X‘05’	X‘01’	X‘1A’(OCL)	964	Large Object Character Locator
X‘05’	X‘01’	X‘1B’(NOCL)	965	Nullable Large Object Character Locator
X‘05’	X‘01’	X‘1C’(OCDL)	968	Large Object Character DBCS Locator
X‘05’	X‘01’	X‘1D’(NOCDL)	969	Nullable Large Obj. Char. DBCS Locator
X‘05’	X‘01’	X‘1E’(RI)	904	Row Identifier
X‘05’	X‘01’	X‘1F’(NRI)	905	Nullable Row Identifier
X‘05’	X‘01’	X‘20’(D)	384	Date
X‘05’	X‘01’	X‘21’(ND)	385	Nullable Date
X‘05’	X‘01’	X‘22’(T)	388	Time
X‘05’	X‘01’	X‘23’(NT)	389	Nullable Time
X‘05’	X‘01’	X‘24’(TS)	392	Timestamp
X‘05’	X‘01’	X‘25’(NTS)	393	Nullable Timestamp
X‘05’	X‘01’	X‘26’(FB)	452	Fixed Bytes
X‘05’	X‘01’	X‘27’(NFB)	453	Nullable Fixed Bytes
X‘05’	X‘01’	X‘28’(VB)	448	Variable Bytes
X‘05’	X‘01’	X‘29’(NVB)	449	Nullable Variable Bytes
X‘05’	X‘01’	X‘2A’(LVB)	456	Long Variable Bytes
X‘05’	X‘01’	X‘2B’(NLVB)	457	Nullable Long Variable Bytes
X‘05’	X‘01’	X‘2C’(NTB)	460	Null-Terminated Bytes
X‘05’	X‘01’	X‘2D’(NNTB)	461	Nullable Null-Terminated Bytes
X‘05’	X‘01’	X‘2E’(NTCS)	460	Null-Terminated SBCS
X‘05’	X‘01’	X‘2F’(NNTCS)	461	Nullable Null-Terminated SBCS
X‘05’	X‘01’	X‘30’(FCS)	452	Fixed Character SBCS
X‘05’	X‘01’	X‘31’(NFCS)	453	Nullable Fixed Character SBCS
X‘05’	X‘01’	X‘32’(VCS)	448	Variable Character SBCS
X‘05’	X‘01’	X‘33’(NVCS)	449	Nullable Variable Character SBCS
X‘05’	X‘01’	X‘34’(LVCS)	456	Long Variable Character SBCS
X‘05’	X‘01’	X‘35’(NLVCS)	457	Nullable Long Variable Character SBCS
X‘05’	X‘01’	X‘36’(FCD)	468	Fixed Character DBCS
X‘05’	X‘01’	X‘37’(NFCD)	469	Nullable Fixed Character DBCS
X‘05’	X‘01’	X‘38’(VCD)	464	Variable Character DBCS
X‘05’	X‘01’	X‘39’(NVCD)	465	Nullable Variable Character DBCS
X‘05’	X‘01’	X‘3A’(LVCD)	472	Long Variable Character DBCS
X‘05’	X‘01’	X‘3B’(NLVCD)	473	Nullable Long Variable Character DBCS
X‘05’	X‘01’	X‘3C’(FCM)	452	Fixed Character Mixed
X‘05’	X‘01’	X‘3D’(NFCM)	453	Nullable Fixed Character Mixed
X‘05’	X‘01’	X‘3E’(VCM)	448	Variable Character Mixed
X‘05’	X‘01’	X‘3F’(NVCM)	449	Nullable Variable Character Mixed
X‘05’	X‘01’	X‘40’(LVCM)	456	Long Variable Character Mixed

Application Class	Meta Data Type	Meta Data Reference DRDA-Type and Name	SQL Data Type	Description
X‘05’	X‘01’	X‘41’(NLVCM)	457	Nullable Long Variable Character Mixed
X‘05’	X‘01’	X‘42’(NTM)	460	Null-Terminated Mixed
X‘05’	X‘01’	X‘43’(NNTM)	461	Nullable Null-Terminated Mixed
X‘05’	X‘01’	X‘44’(PLB)	476	Pascal L String Bytes
X‘05’	X‘01’	X‘45’(NPLB)	477	Nullable Pascal L String Bytes
X‘05’	X‘01’	X‘46’(PLS)	476	Pascal L String SBCS
X‘05’	X‘01’	X‘47’(NPLS)	477	Nullable Pascal L String SBCS
X‘05’	X‘01’	X‘48’(PLM)	476	Pascal L String Mixed
X‘05’	X‘01’	X‘49’(NPLM)	477	Nullable Pascal L String Mixed
X‘05’	X‘01’	X‘4C’(DLS)	396	SBCS Datalink
X‘05’	X‘01’	X‘4D’(NDLS)	397	Nullable SBCS Datalink
X‘05’	X‘01’	X‘4E’(DLM)	396	Mixed-byte Datalink
X‘05’	X‘01’	X‘4F’(NDLM)	396	Nullable Mixed-byte Datalink
X‘05’	X‘01’	X‘C8’(OB)	404	Large Object Bytes
X‘05’	X‘01’	X‘C9’(NOB)	405	Nullable Large Object Bytes
X‘05’	X‘01’	X‘CA’(OCS)	408	Large Object Character SBCS
X‘05’	X‘01’	X‘CB’(NOCS)	409	Nullable Large Object Character SBCS
X‘05’	X‘01’	X‘CC’(OCD)	412	Large Object Character DBCS
X‘05’	X‘01’	X‘CD’(NOCD)	413	Nullable Large Object Character DBCS
X‘05’	X‘01’	X‘CE’(OCM)	408	Large Object Character Mixed
X‘05’	X‘01’	X‘CF’(NOCM)	409	Nullable Large Object Character Mixed

Note: Multiple DRDA types can correspond to the same SQL data type. For example, the DRDA types for FB, FCS, and FCM all correspond to SQL type 452.

Table 5-13 MDD References for Early Group Data Units

Application Class	Meta Data Type	Meta Data Reference DRDA-Type	Data Unit Name	Description
X‘05’	X‘02’	X‘50’	SQLDAGRP	SQL Data Area group description
X‘05’	X‘02’	X‘51’	SQLUDTGRP	SQL User-defined Data Group Description
X‘05’	X‘02’	X‘52’	SQLCAXGRP	SQL Communication Area Exceptions group
X‘05’	X‘02’	X‘54’	SQLCAGRP	SQL Communication Area group description
X‘05’	X‘02’	X‘56’	SQLPAGRP	SQL Privileges Area group description
X‘05’	X‘02’	X‘58’	SQLNUMGRP	SQL Number of Elements group description

Application Class	Meta Data Type	Meta Data Reference DRDA-Type	Data Unit Name	Description
X‘05’	X‘02’	X‘5A’	SQLOBJGRP	SQL Object Name group description
X‘05’	X‘02’	X‘5B’	SQLCIGRP	SQL Result Set Column Information group description
X‘05’	X‘02’	X‘5C’	SQLSTTGRP	SQL Statement group description
X‘05’	X‘02’	X‘5E’	SQLVRBGRP	SQL Statement Variables group description
X‘05’	X‘02’	X‘5F’	SQLRSGRP	SQL Result Set group description

Table 5-14 MDD References for Early Row Descriptors

Application Class	Meta Data Type	Meta Data Reference DRDA-Type	Data Unit Name	Description
X‘05’	X‘03’	X‘60’	SQLDAROW	SQL Data Area row description
X‘05’	X‘03’	X‘64’	SQLCARD	SQL Communication Area row description
X‘05’	X‘03’	X‘66’	SQLPAROW	SQL Privileges Area Repeating group row description
X‘05’	X‘03’	X‘68’	SQLNUMROW	SQL Number of Elements row description
X‘05’	X‘03’	X‘6A’	SQLOBJNAM	SQL Object Name row description
X‘05’	X‘03’	X‘6B’	SQLCIROW	SQL Result Set Column Information row description
X‘05’	X‘03’	X‘6C’	SQLSTT	SQL Statement row description
X‘05’	X‘03’	X‘6E’	SQLVRBROW	SQL Statement Variables row description
X‘05’	X‘03’	X‘6F’	SQLRSROW	SQL Result Set row description

Table 5-15 MDD References for Early Array Descriptors

Application Class	Meta Data Type	Meta Data Reference DRDA-Type	Data Unit Name	Description
X‘05’	X‘04’	X‘74’	SQLDARD	SQLCA and SQLDA array description

Application Class	**Meta Data Type**	**Meta Data Reference DRDA-Type**	**Data Unit Name**	**Description**
X‘05’	X‘04’	X‘78’	SQLPARD	SQLCA and SQLPA array description
X‘05’	X‘04’	X‘7B’	SQLCINRD	SQL Result Set Column Information array description
X‘05’	X‘04’	X‘7E’	SQLSTTVRB	SQL Statement Variables array description
X‘05’	X‘04’	X‘7F’	SQLRSLRD	SQL Result Set array description

Table 5-16 MDD References Used in Late Environmental Descriptors

Application Class	**Meta Data Type**	**Meta Data Reference DRDA-Type and Name**	**SQL Data Type**	**Description**
X‘05’	X‘01’	***	******	Same values as allowed for Early Environmental Descriptors

Table 5-17 MDD References for Late Group Data Units

Application Class	**Meta Data Type**	**Meta Data Reference DRDA-Type**	**Data Unit Name**	**Description**
X‘05’	X‘02’	X‘D0’	SQLDTAGRP	SQL Data Value group description

Table 5-18 MDD References for Late Row Descriptors

Application Class	**Meta Data Type**	**Meta Data Reference DRDA-Type**	**Data Unit Name**	**Description**
X‘05’	X‘03’	X‘E0’	SQLCADTA	Row description for one row with SQLCA and data
X‘05’	X‘03’	X‘E4’	SQLDTA	Row description for one data row

Table 5-19 MDD References for Late Array Descriptors

Application Class	Meta Data Type	Meta Data Reference DRDA-Type	Data Unit Name	Description
X‘05’	X‘04’	X‘F0’	SQLDTARD	SQLCA and Data array definition
X‘05’	X‘04’	X‘F4’	SQLDTAMRW	Row description for multi-row data

The MDDs in the previous tables show DRDA Types. The SDA, GDA, or RLO that follows the MDD is the representation of that type. For ease of use, the standard DRDA descriptor examples have LIDs equal to their DRDA types. This is not a permanent relationship. The relationship exists for early descriptors only. When late environmental descriptors are required, this relationship does not hold. The DRDA semantics are represented in the MDD. The MDD value (DRDA type) should not be inferred from the LID of the descriptor that follows it.

5.7.1 Overriding Descriptors to Handle Problem Data

Descriptors are overridden using two distinct and interacting methods in DRDA.

- The first method overrides environmental specifications originally established at the time a conversation is initiated. This is accomplished with TYPDEFNAM and TYPDEFOVR specifications associated with the data that does not conform to the current specification. This is a global method and can override environmental definitions for everything. See Table 5-20 on page 251 for explanation.
- The second method provides specific field level overrides for user data that does not conform to the TYPDEFNAM and TYPDEFOVR specifications currently in effect. These overrides are accomplished by specification of MDD / SDA pairs of FD:OCA triplets for each class of user data that must be handled. The grouping triplets then refer to the new (special) SDAs to specify the actual representation of the user’s data.

Detail concerning each method follows.

5.7.1.1 Overriding Everything

The example below shows the sequence of FD:OCA triplets that participate in the definition of DRDA data in DRDA Level 3 and before.

```
 Early                               Late
.................................   ..................................................
Environmental   Grp Row Arr         Environmental                        Grp Row Arr
TTTTTTTTTTTTT   MMMMMMMMMMM         OOOOOOOOOOOOOOOOOOOOOOOOOO           UUUUUUUUUUU
```

All early descriptor triplets are grouped together and all late descriptor triplets are grouped together.

The example below shows the sequence of FD:OCA triplets that participate in the definition of DRDA data in DRDA Level 4 and higher.

Early		Late	Early	Late
Environmental	Grp Row Arr	Environmental	Environmental	Grp Row Arr
TTTTTTTTTTTTT	MMMMMMMMMMM	OOOOOOOOOOOOO	TTTTTTTTTTTTT	UUUUUUUUUUUUU

Some early descriptor triplets are taken from the high-end of the late environmental descriptor range to accommodate additional DRDA data types added to DRDA Level 4. They are treated exactly like other early descriptor triplets at the low-end of the early environmental descriptor range.

The early descriptor triplets are broken into two groups: the T triplets and the M triplets. The T triplet values establish the basic representations for all DRDA data. They are established by specifying TYPDEFNAM and/or TYPDEFOVR. The M triplets define DRDA information units (such as SQLCA). They are established with the MGRLVL parameter on EXCSAT.

The T triplet values can be overridden for any command or reply by specifying a new value for TYPDEFNAM or TYPDEFOVR. The override is effective for the life of the command or reply and applies to all DRDA data not subsequently overridden. (See Section 5.7.1.2 on page 252.)

In some cases, TYPDEFNAM and TYPDEFOVR can be specified to override the representation specification provided on the earlier ACCRDB command. They are effective until the end of the command or until overridden again.

Table 5-20 illustrates the cases:

Table 5-20 TYPDEFNAM and TYPDEFOVR

Condition	**Description in Effect for SQLSTT**	**Description in Effect for SQLSTTVRB**
Not supplied	ACCRDB	ACCRDB
Supplied only before SQLSTT	Override Supplied before SQLSTT	Override Supplied before SQLSTT
Supplied only before SQLSTTVRB	ACCRDB	Override Supplied before SQLSTTVRB
Supplied both places	Override supplied before SQLSTT	Override Supplied before SQLSTTVRB

The M triplet values cannot be overridden. These are all grouping and structuring triplets. Any changes to these would mean a change in what information was exchanged rather than just how that information would be represented.

The T and M triplets persist across and throughout a connection to a relational database. Overrides to these triplets and the O and U triplets persist only for the processing of one command or reply.

Similarly, the late descriptor triplets are broken into two groups: the O triplets and the U triplets. The O triplets provide specific overrides and are described in Section 5.7.1.2 on page 252. The U triplets define actual user data, sometimes in combination with DRDA information units. The U triplets reference O triplets and both T triplets and M triplets (which in turn reference T triplets). Data described through the T and M triplets is affected by specification of TYPDEFNAM and TYPDEFOVR.

5.7.1.2 Overriding Some User Data

The key to overriding the representation specification for some or all user data without affecting the rest of the user data and the DRDA information units lies within the override or O triplets. These triplets are placed between the M triplets (which describe DRDA information units) and the U triplets (which describe user data). Based on FD:OCA referencing rules, the U triplets can reference the O triplets and thus provide special representations for user data. The M triplets, however, cannot reference to the right, and, therefore, all the DRDA-defined early information units are bound only to themselves and the T triplets.

MDD / SDA triplet pairs are provided for each class (such as Fixed-Length Character Strings with Single Byte Characters) of user data that must be overridden. The SDA triplets are then referred to appropriately by the grouping triplet to include the field in the data definition and to assign length values as needed. The MDD triplet defines what sort of data is being defined in the DRDA sense. The following SDA triplet describes the pattern of bits that will be used to represent the data.

The TYPDEFNAM and TYPDEFOVR parameters have no effect on the O triplets. For example, if CCSID 437 is specified in an O triplet, then the data must be in CCSID 437 independent of whatever TYPDEFOVR parameter had been specified previously.

5.7.1.3 Assigning LIDs to O Triplets

There are only two considerations. First, stay within a range of 1 to 255, and second, select a LID that does not interfere with references to the early triplets or other O triplets.

The example below shows the LID ranges used by this level of DRDA. Use this only as a guide. These LID assignments are *not* fixed for all time. What is fixed is that the O triplets will never overlap the M and T triplets, and, therefore, O triplet LIDs that match M or T triplet LIDs will block reference to those triplets (SDAs, GDAs, or RLOs).

This example shows the sequence of FD:OCA triplets that participate in the definition of DRDA data for DRDA Level 3 and lower.

```
 Early                                         Late
....................................          .........................................................
Environmental   Grp Row Arr                   Environmental                                Grp Row Arr

TTTTTTTTTTTTT   MMMMMMMMMMMM                  OOOOOOOOOOOOOOOOOOOOOOOOOOO                  UUUUUUUUUUU
01------------4F 5x       6x       7x         80------------------------ ----------------------CF   Dx     Ex     Fx
```

In DRDA Level 4, the example is modified to reflect the fact that some LIDs in the late environmental data range have been used for early descriptors for additional DRDA data types. The example of LID assignments for DRDA Level 4 is as follows:

```
 Early                                       Late                  Early                 Late
...................................        ......................  ...................  ..................
Environmental   Grp Row Arr                Environmental           Environmental        Grp Row Arr

TTTTTTTTTTTTT   MMMMMMMMMMMM               OOOOOOOOOOOOO           TTTTTTTTTTTTT        UUUUUUUUUUU
01------------4F 5x      6x      7x        80-----------------C7   C8--------------CF   Dx    Ex    Fx
```

Observations:

1. The O triplet LIDs have space reserved for them according to the DRDA level in the example shown above. These are:
   - X‘80’ to X‘CF’ in DRDA Level 3 and lower
   - X‘80’ to X‘C7’ in DRDA Level 4 and higher

If assignments are restricted to this range, no conflicts will occur. This range provides LIDs that can be used without concern for conflict.

2. O triplets (like T triplets) are not length-specific and can be reused for several fields of user data. All character fields of the same style and CCSID can refer to the same O triplet with length specification being tailored for each field with the GDA in the late group descriptor.

3. References to triplets are resolved one triplet at a time. In DRDA terms that means that all of the triplets referenced from late group descriptors are resolved before any of the late row descriptor references, and so on.

   This fact allows any of the LIDs to the right of the late group descriptors to be used for late environmental descriptors. This also allows reuse of LIDs assigned by DRDA to late row or late array descriptors. This provides 32 more LIDs that can be used without consideration of what the user's data looks like.

4. If more override LIDS (more than 112 in DRDA Level 3 and below or more than 104 in DRDA Level 4 and above) are required, specific user data must be examined. In addition, the FD:OCA rules must be used that state that LID references are resolved to the first LID that matches to the left of or earlier than the referencing triplet. Duplicates are legal.

   Once an LID is selected for an O triplet, any triplet to the left of that O triplet with the same LID will be inaccessible by triplets to the right of that O triplet.

   However, for important cases, indirect reference through M triplets solves this. Assume, for example, there is some user data where all the user 4-byte integer fields are byte reversed, but the DRDA information units (such as the SQLCA) has integer fields in the normal sequence. If X'02' is selected as the LID for the O triplet to specify this, no late group descriptor (for example, no user data) could reference the regular 4-byte integer format for the environment. However, the U triplets that define the user data will reference M triplets to include DRDA information units. The first match to the left of the M triplet will produce the normal environment's integer. Thus, for some of the data that will flow (the SQLCA) LID X'02' will mean regular sequence and for other data (the user's data) it will mean byte-reversed.

Using all these methods in combination, 250 unique LID values can be approached for O triplets.

5.7.2 MDD Materialization Rules

As shown for each of the specific definitions of triplets for DRDA types, each representation is really a pair of triplets; an MDD that states the type followed by another triplet that states how it is represented.

Section 5.2.3 on page 143 described several cases for which descriptors were required to accompany DRDA data. In some cases, no descriptor information flowed and in others the late descriptors flowed. This section further defines when MDD triplets must be included in late descriptors, and when they can be omitted.

MD-1 Late descriptors that contain No Override Triplets can be built with no MDD triplets. The receiver of the descriptors understands the descriptor format (the sequence of triplets) for each command. DRDA has fixed these formats.

MD-2 Each Late Environmental triplet must be preceded by an MDD triplet that specifies its DRDA type. All Override Triplets require preceding MDDs.

MD-3 Any descriptor that contains an MDD triplet must have an MDD triplet specification for every other triplet to the right of the first MDD. If Override Triplets are provided (these require an MDD), then the subsequent group, row, and array triplets must also be preceded by MDDs that define their types.

A simplified restatement of these rules is that if Override Triplets are required, then every triplet in the late descriptor requires a corresponding MDD; otherwise, no MDD triplets are required.

The use of TYPDEFNAM and TYPDEFOVR specifications does not force the use of MDDs in any late descriptors.

5.7.3 Error Checking and Reporting for Descriptors

Both FD:OCA and DRDA define error conditions. However, this volume defines all possible FD:OCA descriptor syntax error conditions for DRDA. Therefore, descriptors need only pass DRDA validity checks. If the receiver of an FDODSC finds it in error, the error must be reported with a DDM message DSCINVRM. If the descriptor passes DRDA validity checks, but the data does not to match the descriptors, the mismatch must be reported with a DDM message DTAMCHRM.

5.7.3.1 General Errors

01 FD:OCA Triplet not used in DRDA descriptors or Type code invalid.

02 Triplet Sequence Error: the two possible sequences are:

1. GDA,(CPT,)RLO<,RLO> <== normal case with no overrides
2.
```
MDD,SDA,(MDD,SDA,)MDD,GDA,(CPT,)\
MDD,RLO<,MDD,RLO>
```

   where () indicates an optional repeating group and <> indicates a field allowed only when arrays are expected.

03 An array description is required, and this one does not describe an array (probably too many or too few RLO triplets).

04 A row description is required, and this one does not describe a row (probably too many or too few RLO triplets).

05 Late Environmental Descriptor just received not supported (probably due to non-support of requested overrides).

06 Malformed triplet; missing required parameter.

07 Parameter value not acceptable.

5.7.3.2 MDD Errors

11 MDD present is not recognized as DRDA Descriptor.

12 MDD Class not recognized as valid DRDA class.

13 MDD type not recognized as a valid DRDA type.

5.7.3.3 SDA Errors

21 Representation incompatible with DRDA type (in prior MDD).

22 CCSID not supported.

5.7.3.4 GDA/CPT Errors

32	GDA references an LID that is not an SDA or GDA.
33	GDA length override exceeds limits.
34	GDA precision exceeds limits.
35	GDA scale > precision or scale negative.
36	GDA length override missing or incompatible with data type.

5.7.3.5 RLO Errors

41	RLO references an LID that is not an RLO or GDA.
42	RLO fails to reference a required GDA or RLO (for example, QRYDSC must include a reference to SQLCAGRP).

5.8 DRDA Examples

This section provides DRDA examples for environmental descriptions and command execution.

5.8.1 Environmental Description Objects

The following is a sample of all the FD:OCA triplets required to specify the representations of every DRDA type for one specific environment, QTDSQL370. As discussed earlier, the early environment descriptor set is determined during the Access RDB phase of communication establishment between requester and server. The early data unit descriptor set is determined during EXCSAT based on the SQLAM's MGRLVL. The late data unit descriptors must be sent over the link as needed to accompany user data.

This example shows all data type and data unit representations. The descriptors shown are for the System 390 environment. Each is contained in a DDM FDODSC object. The task to construct an equivalent descriptor set for any other environment is straightforward. Just take all the values listed in the boxes for that environment and construct the table.

The descriptors are divided into three groups based on when they are agreed to: Early Environmental, Early Data, or Late Data (ACCRDB, EXCSAT, or user data transfer).

The FDODSC entry in Table 5-21 is a different format than the rest of the table (and the headings). However, it is included to illustrate the assembly of the complete descriptor.

5.8.1.1 Early Environmental Descriptors

The Early Environmental Descriptors in Table 5-21 apply for SQLAM Level 3, SQLAM Level 4, and SQLAM Level 5. The only exceptions are RSL and NRSL, which are not supported at SQLAM Level 3 and SQLAM Level 4.

Table 5-21 Complete set of Early Environmental Descriptors for QTDSQL370

DRDA Type	MDD Descriptor—HEX	SDA, GDA, CPT, or RLO Descriptor—HEX
FDODSC	name=QTDSQL370	03860010 (Descriptor Object)
I4	07780005 010102	0C700223 00000000 00000004
NI4	07780005 010103	0C7003A3 00000000 00000004
I2	07780005 010104	0C700423 00000000 00000002
NI2	07780005 010105	0C7005A3 00000000 00000002
I1	07780005 010106	0C700623 00000000 00000001
NI1	07780005 010107	0C7007A3 00000000 00000001
BF16	07780005 010108	0C700840 00000000 00000010
NBF16	07780005 010109	0C7009C0 00000000 00000010
BF8	07780005 01010A	0C700A40 00000000 00000008
NBF8	07780005 01010B	0C700BC0 00000000 00000008
BF4	07780005 01010C	0C700C40 00000000 00000004
NBF4	07780005 01010D	0C700DC0 00000000 00000004
FD	07780005 01010E	0C700E30 00000000 00001F1F
NFD	07780005 01010F	0C700FB0 00000000 00001F1F
ZD	07780005 010110	0C701033 00000000 00001F1F
NZD	07780005 010111	0C7011B3 00000000 00001F1F
N	07780005 010112	0C701232 000001F4 01001F1F

DRDA Type	MDD Descriptor—HEX	SDA, GDA, CPT, or RLO Descriptor—HEX
NN	07780005 010113	0C7013B2 000001F4 01001F1F
RSL	07780005 010114	0C701423 00000000 00000004
NRSL	07780005 010115	0C7015A3 00000000 00000004
I8	07780005 010116	0C701623 00000000 00000008
NI8	07780005 010117	0C7017A3 00000000 00000008
OBL	07780005 010118	0C701801 00000000 00000004
NOBL	07780005 010119	0C701981 00000000 00000004
OCL	07780005 01011A	0C701A01 00000000 00000004
NOCL	0778000 01011B	0C701B81 00000000 00000004
OCDL	07780005 01011C	0C701C01 00000000 00000004
NOCDL	07780005 01011D	0C701D81 00000000 00000004
RI	07780005 01011E	0C701E02 00000000 00010028
NRI	07780005 01011F	0C701F82 00000000 00010028
D	07780005 010120	0C702010 000001F4 0100000A
ND	07780005 010121	0C702190 000001F4 0100000A
T	07780005 010122	0C702210 000001F4 01000008
NT	07780005 010123	0C702390 000001F4 01000008
TS	07780005 010124	0C702410 000001F4 0100001A
NTS	07780005 010125	0C702590 000001F4 0100001A
FB	07780005 010126	0C702601 00000000 00007FFF
NFB	07780005 010127	0C702781 00000000 00007FFF
VB	07780005 010128	0C702802 00000000 00017FFF
NVB	07780005 010129	0C702982 00000000 00017FFF
LVB	07780005 01012A	0C702A02 00000000 00017FFF
NLVB	07780005 01012B	0C702B82 00000000 00017FFF
NTB	07780005 01012C	0C702C03 00000000 00017FFF
NNTB	07780005 01012D	0C702D83 00000000 00017FFF
NTCS	07780005 01012E	0C702E14 000001F4 01017FFF
NNTCS	07780005 01012F	0C702F94 000001F4 01017FFF
FCS	07780005 010130	0C703010 000001F4 01007FFF
NFCS	07780005 010131	0C701990 000001F4 01007FFF
VCS	07780005 010132	0C703211 000001F4 01017FFF
NVCS	07780005 010133	0C703391 000001F4 01017FFF
LVCS	07780005 010134	0C703411 000001F4 01017FFF
NLVCS	07780005 010135	0C703591 000001F4 01017FFF
FCD	07780005 010136	0C703610 0000012C 02003FFF
NFCD	07780005 010137	0C703790 0000012C 02003FFF
VCD	07780005 010138	0C703811 0000012C 02013FFF
NVCD	07780005 010139	0C703991 0000012C 02013FFF
LVCD	07780005 01013A	0C703A11 0000012C 02013FFF
NLVCD	07780005 01013B	0C703B91 0000012C 02013FFF
FCM	07780005 01013C	0C703C10 000003A2 01007FFF
NFCM	07780005 01013D	0C703D90 000003A2 01007FFF
VCM	07780005 01013E	0C703E11 000003A2 01017FFF
NVCM	07780005 01013F	0C703F91 000003A2 01017FFF
LVCM	07780005 010140	0C704011 000003A2 01017FFF
NLVCM	07780005 010141	0C704191 000003A2 01017FFF

DRDA Type	MDD Descriptor—HEX	SDA, GDA, CPT, or RLO Descriptor—HEX
NTM	07780005 010142	0C704214 000003A2 01017FFF
NNTM	07780005 010143	0C704394 000003A2 01017FFF
PLB	07780005 010144	0C704407 00000000 000100FF
NPLB	07780005 010145	0C704587 00000000 000100FF
PLS	07780005 010146	0C704619 000001F4 010100FF
NPLS	07780005 010147	0C704799 000001F4 010100FF
PLM	07780005 010148	0C704819 000003A2 010100FF
NPLM	07780005 010149	0C704999 000003A2 010100FF
DLS	07780005 01014C	0C704C11 000001F4 01017FFF
NDLS	07780005 01014D	0C704D91 000001F4 01017FFF
DLM	07780005 01014E	0C704E11 000003A2 01017FFF
NDLM	07780005 01014F	0C704F91 000003A2 01017FFF
OB	07780005 0101C8	0C70C850 00000000 00018008
NOB	07780005 0101C9	0C70C9D0 00000000 00018008
OCS	07780005 0101CA	0C70CA51 000001F4 01018008
NOCS	07780005 0101CB	0C70CBD1 000001F4 01018008
OCD	07780005 0101CC	0C70CC51 0000012C 02018008
NOCD	07780005 0101CD	0C70CDD1 0000012C 02018008
OCM	07780005 0101CE	0C70CE51 000003A2 01018008
NOCM	07780005 0101CF	0C70CFD1 000003A2 01018008

5.8.1.2 *Early Data Unit Descriptors*

The Early Data Unit Descriptors in Table 5-22 apply for SQLAM Level 3, SQLAM Level 4, and SQLAM Level 5. The only exceptions are SQLRSGRP, SQLRSROW, SQLRSLRD, SQLCIROW, SQLCIGRP, and SQLCINRD, which are not supported at SQLAM Level 3 and SQLAM Level 4.

Table 5-22 Complete set of Early Data Unit Descriptors

DRDA Type	MDD Descriptor—HEX	SDA, GDA, CPT, or RLO Descriptor—HEX
FDODSC	unnamed	00F80010 (Descriptor Object)
SQLDAGRP	07780005 020150	24755004 00020400 02020004 04000226 00023E00 1E32001E 3E001E32 001E3E00 FE3200FE
SQLCAXGRP	07780005 020152	3F765230 00120200 04020004 02000402 00040200 04020004 30000130 00013000 01300001 30000130 00013000 01300001 30000130 00013000 013E0046 320046
SQLCAGRP	07780005 020154	0F765402 00043000 05300008 520000
SQLPAGRP	07780005 020156	1B75563E 00123200 123E0012 32001230 00013000 01300001 300019
SQLNUMGRP	07780005 020158	06755804 0002
SQLOBJGRP	07780005 02015A	09755A3E 00FE3200 FE
SQLCIGRP	07780005 02015B	15755B3E 001E3200 1E3E001E 32001E3E 00FE3200 FE

DRDA Type	MDD Descriptor—HEX	SDA, GDA, CPT, or RLO Descriptor—HEX
SQLSTTGRP	07780005 02015C	09755C40 7FFF347 FF
SQLVRBGRP	07780005 02015E	1E755E04 00020400 02020004 04000226 00023E00 40320040 E3004032 0040
SQLRSGRP	07780005 02015F	0F755F14 00043E00 1E32001E 020004
SQLDAROW	07780005 030160	06716050 0001
SQLCARD	07780005 030164	06716454 0001
SQLPAROW	07780005 030166	06716656 0001
SQLNUMROW	07780005 030168	06716858 0001
SQLOBJNAM	07780005 03016A	06716A5A 0001
SQLCIROW	07780005 03016B	06716B5B 0001
SQLSTT	07780005 03016C	06716C5C 0001
SQLVRBROW	07780005 03016E	06716E5E 0001
SQLRSROW	07780005 03016F	06716F5F 0001
SQLDARD	07780005 040174	0C717464 00016800 01600000
SQLPARD	07780005 040178	0C717864 00016800 01660000
SQLCINRD	07780005 04017B	09717B68 00016B00 00
SQLSTTVRB	07780005 04017E	09717E68 00016E00 00
SQLRSLRD	07780005 04017F	09717F68 00016F00 00

5.8.1.3 Late Data Unit Descriptors

The Late Data Unit Descriptors in Table 5-23 apply for both SQLAM Level 3 and SQLAM Level 4. The only exception is SQLDTAMRW, which is not supported at SQLAM Level 3.

Table 5-23 Complete Set of Late Data Unit Descriptors

DRDA Type	MDD Descriptor—HEX	SDA, GDA, CPT, or RLO Descriptor—HEX
FDODSC	unnamed	LLLL0010 (Descriptor Object)
SQLDTAGRP	07780005 0201D0	. . 76D0. . . 7F00.. ....
SQLCADTA	07780005 0301E0	0971E054 0001D000 01
SQLDTA	07780005 0301E4	0671E4D0 0001
SQLDTARD	07780005 0401F0	0671F0E0 0000
SQLDTAMRW	07780005 0401F4	0671F4E4 0000

5.8.2 Command Execution Examples

The following examples of DRDA command execution illustrate how the descriptors would be assembled to produce actual flows.

These examples use Table 5-24 on page 260, which is resident in a QTDSQL370 machine and is called STATS.

Table 5-24 STATS Sample Table

AGE SMALLINT Nullable	WEIGHT SMALLINT Nullable	NAME VARCHAR(20) Not Null
21	160	BOB
30	190	JIM
35	180	SAM
25	170	JOE
40	150	ROD

These examples assume that the application requester prefers the QTDSQLX86 environment.

5.8.2.1 EXECUTE IMMEDIATE

This is the SQL statement for the first example:

```
EXEC  SQL  EXEC  IMMEDIATE  ’GRANT  SELECT  ON  STATS  TO  BRUCE’
```

Because this is an EXECUTE IMMEDIATE command, the application requester sends the statement to the application server using DDM’s EXCSQLIMM. Table 5-1 on page 146 shows that command data flows according to early descriptor SQLSTT and that reply data will always be an SQLCA.

The actual bytes that flow to show this data are in Table 5-25. This table does not show the DDM command proper and its parameters.

Table 5-25 EXECUTE IMMEDIATE Command Data

Reference	Hex Representation	Description
SQLSTT	00262414	DDM length and code point for SQL Statement
SQLSTT	001E 4752 414E5420 53454C45 4354204F 4E205354 41545320 544F2042 52554345 0000	30 (Length of Statement) GRANT SELECT ON STATS TO BRUCE 00

The length of the variable-length field is not byte reversed, but all the characters are sent in the application requester’s preferred code (ASCII).

After the application server processes it, the application requester expects an SQLCA in response. If it worked as expected, it would have an SQLCODE of 0 (SQLSTATE ’00000’). (See Table 5-26.)

Table 5-26 EXECUTE IMMEDIATE Reply Data

Reference	Hex Representation	Description
SQLCARD	00052408	DDM code point for SQLCARD
SQLCARD	FF	Null SQLCARD—all OK

5.8.2.2 *Open Query Statement*

These are the SQL statements for this example (in PL/I):

```
EXEC  SQL  DECLARE  mycursor  CURSOR  FOR
      SELECT  *  FROM  STATS  WHERE  WEIGHT  >  :WGT;
EXEC  SQL  OPEN  mycursor;
EXEC  SQL  FETCH  mycursor
      INTO  :VAGE:VAGEI,  :VWGT:VWGTI,  :VNAME;
```

Variable WGT has been declared as FLOAT(8) and has the value 175.07. Variable VNAME has been declared as CHARACTER VARYING (30). All other variables have been declared as FIXED(15).

This example shows execution of an Open Query request. The statement is previously bound and the request to execute is sent from the application requester to the application server using DDM's OPNQRY command. Table 5-1 on page 146 shows that for OPNQRY command data flows according to late descriptor SQLDTA and that reply data will be an SQLCARD (for error cases) or data that the late descriptor SQLDTARD described.

Table 5-27 shows the actual bytes that flow to show the command data. It does not show the DDM command proper and its parameters.

Table 5-27 Open Query Command Data

Reference	Hex Representation	Description
OBJDSS	0027D003 xxxx	Object Data Stream Structure
SQLDTA	00212412	DDM code point for SQL objects with FD:OCA Descriptors and Data
FDODSC	00100010	DDM code point for FD:OCA Descriptor objects **Note:** MDD/SDA pairs for unusual data would be here if they were required. Also the presence of MDD/SDA pairs here would force inclusion of MDDs before each GDA and RLO that follows.
SQLDTAGRP	0676D0	Start Nullable Group Descriptor—GDA Header
SQLDTAGRP	0A0008	Continue—One Eight-Byte Float Field
SQLDTA	0671E4	Start Row Descriptor—RLO Header
SQLDTA	D00001	Continue—One occurrence of all elements of group X'D0', user data
FDODTA	000D147A	DDM code point for FD:OCA Data objects
FDODTA	000AD7A3 703DE265 40	The data—175.07 (in a nullable group)

The application requester sent the data as FLOAT(8) even though the table column being compared was SMALLINT. The application requester also sent the data in its preferred format, byte reversed. The database manager at the application server end does the conversion based on the SQLDA that describes the input data.

After the application server processes the data, the application requester expects to see a description of the data being returned and the data from the table. In addition, the application server must handle all situations in which an error from the relational database can be reported as a warning in the manner that produces the warning. The application requester is then responsible for upgrading the warning to an error if the application has not made the request in the manner that allows the warning to be passed. See Section 7.17 on page 306 for a description of these responsibilities. If it worked as expected, it returns the descriptor, two rows of data, the End of Query Reply Message, and an End of File SQLCA.

Table 5-28 Open Query Reply Data

Reference	Hex Representation	Description
RPYDSS	0016D052 xxxx	Reply Data Stream Structure
OPNQRYRM	00102205 00061149 00000006 21022417	Open Query Reply Message
OBJDSS	0043D053 xxxx	Object Data Stream Structure
QRYDSC	001F241A	DDM code point for FD:OCA Descriptor objects **Note:** MDD / SDA pairs for unusual data would be here if they were required.
SQLDTAGRP	0C76D0	Start Nullable Group Descriptor—GDA Header
SQLDTAGRP	05000205 00023200 14	Continue—Two nullable SMALLINT and one VARCHAR(20) field
SQLCADTA	0971E0	Start Row Descriptor—RLO Header X‘E0’
SQLCADTA	540001	Continue—One occurrence of all elements of group X‘54’, SQLCA
SQLCADTA	D00001	Continue—One occurrence of all elements of group X‘D0’, User Data
SQLDTARD	0671F0	Start Array Descriptor—RLO Header
SQLDTARD	E00000	Continue—All occurrences of all elements of row X‘E0’, SQLCA with user data
QRYDTA	001E241B	DDM code point for FD:OCA Data objects
QRYDTA	FF000000 230000B4 0003E2C1 D4	First Row—null SQLCA, 35, 180, SAM(3)
QRYDTA	FF000000 1E0000BE 0003D1C9 D4	Second Row—null SQLCA, 30, 190, JIM(3)
RPYDSS	0010D052 xxxx	Reply Data Stream Structure
ENDQRYRM	000A220B 00061149 0004	End of Query Reply Message
OBJDSS	001DD003 xxxx	Object Data Stream Structure
SQLCARD	00172408	DDM code point for the SQLCARD (stand alone)
SQLCARD	00000000 64F0F2F0 F0F0C4E2 D5E7D9C6 C3C8FF	EOF SQLCA (SQLCODE, SQLSTATE and SQLERRPROC only)

The EOF SQLCA becomes null after SQLERRPROC because of the presence of the nullable group SQLCAXGRP inside the SQLCARD.

The entire result was short enough to be included in the first block of data returned as a result of the command. The program has to issue three fetches subsequent to the Open to get all of the data and the EOF indicator. This data flows in DDM OBJDSSs and RPYDSSs.

Because EOF was reached within this block for the OPNQRY command, an ENDQRYRM indicating that the cursor has been closed followed this object. On the third fetch, the application requester receives the ENDQRYRM and the SQLCA. The application requester can then respond to that fetch with the EOF SQLCA and give an SQLSTATE X‘00000’ SQLCA to the Close Cursor request when the application issues it.

5.8.2.3 Insert (Multi-Row)

These are the SQL statements for this example (in PL/I):

```
EXEC  SQL  INSERT  INTO  STATS  :NBR
ROWS  VALUES  (:NEWENTS)
```

Variable NBR is declared as a one-byte integer and has the value 2. Variable NEWENTS is declared as an array of dimension 2. The array structure matches the columns of the STATS table and has the following values:

```
40    170    ROBERT
25    160    STEVE
```

This example shows execution of a multi-row insert request. The INSERT statement was previously bound, and the request to execute is sent from the application requester to the application server using DDM's EXCSQLSTT command.

Table 5-29 shows the actual bytes that flow to show the command data. It does not show the DDM proper command and its parameters.

Table 5-29 Multi-Row Insert Command Data

Reference	Hex Representation	Description
OBJDSS	0047D003 xxxx	Object Data Stream Structure
SQLDTA	00412412	DDM code point for SQL objects with FD:OCA descriptors and data
FDODSC	001C0010	DDM code point for FD:OCA descriptor objects
SQLDTAGRP	0C76D0	Start Nullable Group Descriptor—GDA Header
SQLDTAGRP	05000205 00023200 14	Continue—Two nullable SMALLINT fields and one VARCHAR(20)
SQLDTA	0671E4	Start Row Descriptor—RLO Header
SQLDTA	D00001	Continue—One occurrence of all elements of group X'D0', user data
SQLDTAMRW	0671F4	Start Array Descriptor—RLO Header
SQLDTAMRW	E40000	Continue—All occurrences of all elements of row X'E4', user data
FDODTA	0021147A	DDM code point for FD:OCA data objects
FDODTA	00000028 0000AA00 06D9D6C2 C5D9E2	First row—40, 170, ROBERT(6)
FDODTA	00000019 0000A000 05E2E3C5 E5C5	Second row—25, 160, STEVE(5)

5.8.2.4 Call (Stored Procedure)

The following example of DRDA command execution illustrates how the descriptors would be assembled to produce actual flows for a CALL statement. The SQL statement for this example is:

```
EXEC  SQL  CALL  RMTPROC
(:VAGE:VAGEI, :VWGT:VWGTI, :VNAME, 'ABC', NULL, USER);
```

In this example the host variables are declared and set as follows:

- VWGT is FLOAT(8) and set to 175.07.
- VNAME is CHARACTER VARYING (20) and set to 'FRED'.
- VAGE is FIXED(15) and is not set.

The host indicator variables are set as follows:

- VAGEI is –1.
- VWGTI is 0.

The modes of all parameters are INPUT except for VAGE, which is OUTPUT.

Table 5-30 shows the data that flows in the OBJDSS (object data stream structure) which follows the EXCSQLSTT command. Notice only host variable parameters flow.

Table 5-31 on page 266 shows an example reply data stream. Notice the use of –128 (X'80') indicator values in the FDODTA to flag the second and third parameters as being INPUT only.

Table 5-30 Object Data Stream Example for Execution of CALL Statement

Reference	Hex Representation	Description
OBJDSS	002C D003 xxxx	Object Data Stream Structure
SQLDTA	0026 2412	DDM Length and code point (LLCP) for SQLDTA
FDODSC	0016 0010	DDM Length and code point (LLCP) for FDODSC
SQLDTAGRP	0C76D0	Start Nullable Group Desc.-GDA header
SQLDTAGRP	050002 0B0008 330014	Continue-SMALLINT, 8 byte FLOAT, VARCHAR(20), all nullable
SQLDTA	0671E4	Start Row Descriptor - RLO Header
SQLDTA	D00001	Continue-One occurrence of all elements of group X'D0', user data
FDODTA	0016147A	DDM Length and code point (LLCP) for FDODTA
FDODTA	00	Non-null nullable group indicator
FDODTA	FF	Null indicator for first parameter
FDODTA	000AD7A3 703DE265 40	Data for second parameter (175.07) in nullable field
FDODTA	000004C6 D9C5C4	Data for third parameter ('FRED') in nullable field

Table 5-31 Reply Data Stream Example for Execution of CALL Statement

Reference	Hex Representation	Description
OBJDSS	0034 D002 xxxx	Object Data Stream Structure
SQLDTARD	002E 2413	DDM Length and code point (LLCP) for SQLDTARD
FDODSC	001F 0010	DDM LLCP for FDODSC
SQLDTAGRP	0C76D0	Start Nullable Group Desc.-GDA header
SQLDTAGRP	050002 0B0008 330014	Continue-SMALLINT, 8 byte FLOAT, VARCHAR(20), all nullable
SQLCADTA	0971E0	Start Row Descriptor - RLO Header X‘E0’
SQLCADTA	540001	Continue-One occurrence of all elements of group X‘54’, SQLCA
SQLCADTA	D00001	Continue-One occurrence of all elements of group X‘D0’, user data
SQLDTARD	0671F0	Start Row Descriptor - RLO Header
SQLDTARD	E00000	Continue-ALL occurrences of all elements of group X‘E0’, SQLCA with user data
FDODTA	000B147A	DDM LLCP for FDODTA
FDODTA	FF	Null SQLCA
FDODTA	00	Non-null nullable group indicator
FDODTA	000020	Non-null first parameter (32) in nullable field
FDODTA	8080	Special INPUT-only null indicator values (–128) for second and third parameters

5.8.2.5 Call (Stored Procedure Returning Result Sets)

The following example illustrates the actual flow for the summary component of the response to an SQL statement that invokes a stored procedure and returns result sets. The example flow is for the summary component of Figure 4-16 on page 101. The example assumes that the RSLSETRM reply message does not contain a server diagnostic information (SRVDGN) reply parameter, that there was no need for the application server to specify TYPDEFNAM and TYPDEFOVR overrides, and that the SQLSTATE for the SQL statement that invoked the stored procedure is X'00000'.

Table 5-32 Reply Data Stream Example for Summary Component of Response

Reference	Hex Representation	Description
RPYDSS	009CD052 xxxx	Reply Data Stream Structure
RSLSETRM	00962219	DDM length and code point (LLCP) for RDB Result Set Reply Message
SVRCOD	00061149 0000	RDB Result Set Reply Message Severity Code
PKGSNLST	008C2139	DDM length and code point (LLCP) for RDB Package Name, Consistency Token, and Section Number List
PKGNAMCSN	00442113 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx	PKGNAMCSN for result set #1
PKGNAMCSN	00442113 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx	PKGNAMCSN for result set #2
OBJDSS	0065D003 xxxx	Object Data Stream Structure
SQLCARD	00052408	DDM length and code point (LLCP) for SQL Communications Area Reply Data
SQLCARD	FF	Null SQLCARD
SQLRSLRD	005A240E	DDM length and code point (LLCP) for SQL Result Set Reply Data
SQLRSLRD	0002	Number of result set entries

Reference	Hex Representation	Description
SQLRSLRD	xxxxxxxx 001Exxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx 0000xxxx xxxx	Locator value, name, and number of rows for result set #1
SQLRSLRD	xxxxxxxx 001Exxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx 0000xxxx xxxx	Locator value, name, and number of rows for result set #2

Chapter 6

Names

For DRDA, many named resources, such as SQL tables, must be uniquely accessible from anywhere within a set of interconnected networks. The names must also be convertible as necessary to addresses or routings in order to complete the connection between the application that needs the data and the database management system that supplies the stored data.

In the DRDA environment, database management systems join networks, networks merge, networks split, data moves from one database management system to another, and programs migrate from one system to another. In short, the DRDA environment is constantly evolving. Careful attention to the naming of users and resources is crucial for success in such a dynamic environment.

A user's identification and the authorities that go along with the ID should not change if the user enters the environment from different machines. For example, all PCs in a pool of LAN-connected PCs should have equivalent access to host data. As data migrates about the set of interconnected networks to help performance or reliability, programs and stored queries should not require modification. Physical changes to data configurations should not directly affect users or programs.

In short, named entities in the DRDA environment need to be identified uniquely in their operating environment. This can include worldwide global uniqueness. Global uniqueness can be achieved through standardized naming structures or name registration organizations or a combination of both.

This chapter describes DRDA naming conventions for:

- Names for end users
  - — The environment defines the allowable name structure.
  - — DRDA places restrictions on the name structure.
  - — The name structure does not guarantee uniqueness. The environment must guarantee uniqueness.
- Names for relational databases
  - — DRDA defines name structure.
  - — RDB_NAME prefix registration allows for global uniqueness.
- Names for tables and views
  - — SQL defines name structure.
  - — Structure allows for global uniqueness.
- Names for packages
  - — SQL defines name structure.
  - — Structure allows for global uniqueness.
- Names for target programs
  - — Target program name structures will be different dependent on the communications environment in effect (See Part 3, Network Protocols.)

See **Referenced Documents** on page xxiv for sources that provide background information for better understanding of this chapter.

6.1 End Users

An end-user name must be unique or uniquely identifiable at the relational database that is being accessed.

6.1.1 Support for End-User Names

DRDA implementations provide the following support for DRDA end user names.

An end-user identification at the application requester consists of a single token that makes the identification unique at one, or possibly more than one, application server.

Syntax

```
USER_ID
    8    (8 bytes total)
```

The character string that represents an end user-name within DRDA flows has a maximum length of 8 bytes and must consist of letters (A through Z) and numerics (0 through 9).

Semantics

USER_ID
Uniquely identifies a user within the scope of the user ID name space of an application server. The application server should attempt to match the USER_ID to the name space of the local security manager. For example, if USER_IDs are required to be in uppercase, USER_IDs should be folded to uppercase before authentication.

The application requester passes the USER_ID to the application server by one of the following methods:

- Through the network protocol; for example, in the LU 6.2 ALLOCATE verb
- Through the security context information passed in a DDM SECTKN object
- Through DDM *usrid* parameter passed on the SECCHK command

This support assumes the following:

- It is the responsibility of the end user to obtain a unique USER_ID at an application server.
- An end user can need a different USER_ID at each application server that contains data the end user desires to access.

6.2 RDBs

The name for a relational database is RDB_NAME.

Syntax

```
RDB_NAME
    18 bytes total
```

An RDB_NAME has the same syntactic constraints as SQL identifiers with the exception that RDB_NAME cannot contain the alphabetic extenders for national languages (#, @, and $, for example). The valid characters are uppercase letters (A through Z), the numerics (0 through 9), and the underscore character (_). The maximum length of an RDB_NAME is 18 bytes.

The description of the syntax of the RDB_NAME does not imply syntax checking is required in DRDA. When the application tries to access the relational database, it finds the invalid RDB_NAMEs. Invalid RDB_NAMEs are based on the non-existence of the RDB_NAMEs and not on their syntax. The syntax of the RDB_NAME should be checked when the relational database is created.

Semantics

RDB_NAME

Identifies a relational database. A relational database consists of a relational database management system catalog and all the relational database objects that the catalog describes, as well as the algorithms that access and manipulate the catalog and database objects that the catalog describes.

Note: The SNA Netid Registry registers the first six bytes of the RDB_NAME. The Open Group submits requests to register the first six bytes of an RDB_NAME to the registrar of the SNA Netid Registry in response to customer requests. For more details on the registration process, contact The Open Group.

6.3 Tables and Views

The globally unique name for a table or view is RDB_NAME.COLLECTION.OBJECTID.

Syntax

```
RDB_NAME.COLLECTION.OBJECTID
   18       18         18    (56 bytes total)
```

Section 6.2 defines the syntax of RDB_NAME. COLLECTION and OBJECTID have the same syntactic constraints as SQL identifiers. COLLECTION and OBJECTID are further restricted to be only in the single-byte character set (SBCS). The maximum length of COLLECTION is 18 bytes. The maximum length of OBJECTID is 18 bytes.

Semantics

RDB_NAME

Identifies the relational database whose catalog contains information for the object. Refer to Section 6.2 on page 271 for further detail.

DRDA requires that an application server support the receipt of RDB_NAME in table and view names. DRDA defines the semantic characteristics of RDB_NAME.

COLLECTION

Identifies a unique collection of objects contained within the relational database that RDB_NAME identifies.

OBJECTID

The combination of COLLECTION and OBJECTID uniquely identifies a table or view within the identified relational database.

6.4 Packages

Each relational database management system provides a program preparation process that prepares an SQL application program for execution.

A package is one of the outputs of applying the program preparation process to an SQL application program. A package consists of sections that bind the SQL statements in an application program to access paths at the relational database management system, which stores the tables that the SQL statements reference. The relational database management system that stores the tables also stores and manages the packages that reference the tables.

The package creation process consists of two logical steps:

- The first step extracts the SQL statements and any associated application variable declarations from the application program and replaces the SQL statements with calls to runtime database programs. In doing so, the first step also generates the runtime structures that the application program passes to the runtime database programs during execution.
- The second step binds the extracted SQL statements to access paths at the relational database management system that stores the tables.

The name of the package relates an application program to its selected access paths. The runtime structures stored in the application program contain part of this name.

6.4.1 Package Name

The fully qualified name for a package, or database management system access module, is RDB_NAME.COLLECTION.PACKAGEID.

Syntax

```
RDB_NAME.COLLECTION.PACKAGEID
   18       18         18    (56 bytes total)
```

Section 6.2 on page 271 defines the syntax of RDB_NAME. COLLECTION and PACKAGEID have the same syntactic constraints as SQL identifiers. COLLECTION and PACKAGEID are further restricted to be in the Single-Byte Character Set (SBCS) only. The maximum length of PACKAGEID is 18 bytes. For more information, see *ISO/IEC 9075: 1992, Database Language SQL.*

Semantics

RDB_NAME
Identifies the relational database that is the application server database manager for the package (such as the relational database where creation of the access module occurs). Refer to Section 6.2 on page 271 for further detail.

COLLECTION
Identifies a unique collection of packages contained within the relational database that RDB_NAME identifies.

PACKAGEID
The combination of COLLECTION and PACKAGEID uniquely identifies a package within the application server relational database.

The bind process provides the RDB_NAME, the COLLECTION, and the PACKAGEID for the fully qualified package name.

6.4.2 Package Consistency Token

Each package also has an associated consistency token. The consistency token uniquely identifies the SQL application program preparation process that prepared the source SQL statements for execution. The relational database management system uses the consistency token during SQL program execution to verify that the package it selects for database management access is the instance of the package that the program preparation process generated for the executing instance of the application program. Both the package name and the consistency token flow at execution time to identify the package and confirm the relationship between the package and the application program.

The first step of the program preparation process (see Section 6.4 on page 273) establishes the consistency token. The first step can either generate the consistency token or receive the consistency token as an input parameter.

Syntax

```
PACKAGE_CONSISTENCY_TOKEN
    8                       (8 bytes total)
```

A consistency token is a byte string of length 8.

Semantics

The consistency token uniquely identifies the SQL application program preparation process that prepared the source SQL statements for execution. As such, it associates an execution instance of an SQL application program with a particular instance of a package.

6.4.3 Package Version ID

In order to support orderly management of SQL application programs, it is necessary to recognize that programs can exist in several versions, that the several versions can exist simultaneously, and that each version will have its own package.

The objective of SQL application program version management is to allow a single SQL application program to exist in multiple versions. All versions share the same identity as the application program but must be distinguishable when the application creates new versions of an SQL program, destroys existing versions, or selects the instance of the application program used in other operations such as compile, link edit, and execute.

DRDA does not define how program management components externalize and support versions of programs. However, because a package is the representation of database management access requests for a version of an application program, DRDA incorporates a mechanism to name versions of a package and to resolve an application program database management access request to the proper version of the package.

DRDA supports SQL application program versions by associating a version ID attribute with a package name that serves as the external identifier of the package. DRDA requires specification of the version ID attribute during the creation or dropping of a package and during the granting and revoking of package execution privileges. When a version of an application program executes, the consistency token that the precompiler assigned is used as the execution time selector of the package version.

The existence of a version ID attribute means that every version of a package has two unique names: the package name plus version ID qualifier and the package name plus consistency token qualifier. Users specify the version ID at the user interfaces. The relational database management system uses the consistency token internally to uniquely identify the correct package version for a particular instance of an SQL application program.

A package can have a null version ID. This means that the package has no versions or that one version of the package is not qualified with an external identifier. A consistency token must exist for each package and must be unique across all versions of a package.

Syntax

```
PACKAGE_VERSION_ID
    254               (254 bytes total)
```

A version ID is a varying-length character string having a maximum length of 254 bytes.

Semantics

The version ID uniquely identifies an instance of a package to users.

6.4.4 Sections

A section number uniquely identifies a section within the referenced package. A section number is a 2-byte signed binary integer. The maximum section number value is 32,767 and the minimum section number value is 1.

During the bind process, the application requester sends a section number with a value between one and MAXSCTNBR to each source SQL statement. MAXSCTNBR is a parameter on the DDM command ENDBND. This parameter is a value greater than zero and less than 32K and allows the relational database to ensure that the package contains the correct number of sections. The program preparation process assigns section numbers in increasing order. Gaps in the sequence (such as unassigned section numbers) are possible. The application server can see the section numbers in the gaps later in the bind process if the section number in the gap was assigned earlier at the application requester for a related SQL statement that did not flow (for example, a Declare Cursor for a previously prepared statement), or might not see the section numbers until they are executed.

The program preparation process at the application requester assigns the same section number to all related SQL statements that have execution time dependencies. In particular, each declared statement or cursor receives a unique section number. A cursor declared for a statement shares the statement section number. And each SQL statement that references the declared statement or cursor (FETCH, EXECUTE, OPEN, CLOSE, PREPARE) receives the same section number as the referenced statement or cursor.

The program preparation process at the application requester assigns a unique section number to each of the following: ALTER, CALL, COMMENT ON, CREATE, DELETE, DROP, EXPLAIN, GRANT, INSERT, LABEL ON, LOCK, REVOKE, SELECT (embedded), SET, and UPDATE. The program preparation process also assigns a unique section number to each incompletely understood SQL statement.

The program preparation process at the application requester can assign one section number to all EXECUTE IMMEDIATEs or more than one section number to the set of EXECUTE IMMEDIATEs.

The bind process at the application server handles each EXECUTE IMMEDIATE with a different section number as an independent statement. In products that generate sections, the bind process at the application server generates a separate section for each EXECUTE IMMEDIATE with a different section number.

All SQL statements in an application program are input to the bind process at the application server with the following exceptions: INCLUDE, WHENEVER, PREPARE, EXECUTE,

EXECUTE IMMEDIATE, DESCRIBE, OPEN, FETCH,[39] CLOSE, COMMIT, CONNECT, ROLLBACK, RELEASE, SET CONNECTION, DISCONNECT, BEGIN DECLARE SECTION, END DECLARE SECTION, and local statements.[40] The application requester processes the excepted statements.

ISO/IEC 9075: 1992, Database Language SQL defines the order in which SQL statements must be input. Aside from the rules already stated, the manner in which the bind process handles an individual SQL statement is specific to the environment.

40. FETCH statements that contain multi-row and scroll parameters flow at bind time to provide the application server optimization hints in regard to cursors that fetch multi-rows and/or scroll. If the application server does not support multi-row and/or scrolling, it must reject the FETCH at bind time so the application is informed of the multi-row and scrolling capabilities at the application server. The flowing of FETCH at bind time is not supported in DRDA Level 1.

40. A local statement is understood by the precompiler and either processed completely by the precompiler, or it results in a call to the application requester at runtime, which does not cause any flows to the application server.

6.5 Stored Procedure Names

The qualified form for a stored procedure name is RDB_NAME.COLLECTION.PROCEDURE.

Syntax

```
RDB_NAME.COLLECTION.PROCEDURE
   18       18         18    (56 bytes total)
```

Section 6.2 on page 271 defines the syntax of RDB_NAME. COLLECTION and PROCEDURE have the same syntactic constraints as SQL identifiers. COLLECTION and PROCEDURE are further restricted to be only in the single-byte character set (SBCS). The maximum length of COLLECTION is 18 bytes. The maximum length of PROCEDURE is 18 bytes.

Semantics

RDB_NAME

Identifies the relational database whose catalog contains information for the procedure. Refer to Section 6.2 on page 271 for further detail.

DRDA requires that an application server support the receipt of RDB_NAME in stored procedure names. DRDA defines the semantic characteristics of RDB_NAME.

COLLECTION

Identifies a unique collection of procedures contained within the relational database that RDB_NAME identifies.

PROCEDURE

The combination of COLLECTION and PROCEDURE uniquely identifies a stored procedure within the identified relational database.

6.6 Synonyms and Aliases

The resolution of synonyms and aliases for DRDA tables and views occurs at the application server for DRDA flows. DRDA, however, does not define the mechanism that resolves synonyms and aliases. The particular resolution mechanisms are specific to the environment.

6.7 Default Mechanisms for Standardized Object Names

Refer to Section 6.3 on page 271 for a discussion of the DRDA-defined default values within DRDA object names.

In general, DRDA does not define the mechanism that provides the default values for components of DRDA table, view, and package names for DRDA flows. The particular mechanisms for providing product default values are implementation-specific.

6.8 Target Program

DRDA requires that an application requester (AR) specify the target program name of the application server (AS) when allocating a network connection. The application requester determines the program name of the application server during the process of resolving the RDB_NAME of the application server to a network location. DRDA allows the use of any valid program name that meets the standards of the communications environment that is in use (see Part 3, Network Protocols) and that the application server supports.

To avoid potential name conflicts, the application server program name should be, but need not be, a registered target program name.

DDM might also provide a registered target program name that can be used. The DDM target program name would be used if the DDM implementation at the application server provided file server functions in addition to DRDA functions.

DRDA defines default target program names. The default target program name must be definable at each location that has an application server providing DRDA capabilities. An application requester can then assume the existence of the default target program name at any location providing DRDA capabilities, and default to a target program name when a request requiring an initialization of a network connection does not specify a target program name. Because target programs can have aliases, the default target program name can also have the DDM default target program name or some other registered DRDA target program name. DRDA, however, does not require that a DRDA target program have multiple target program names.

See the following sections for an interpretation of target program names per environment:

- Section 12.8.3 on page 419
- Section 13.6.3 on page 436

Chapter 7

DRDA Rules

This chapter consists of a topical collection of all the rules pertaining to DRDA usage. These rules have been either described, alluded to, implied, or referenced in other chapters of the DRDA reference.

The major exception to the collection of rules is the omission of architecture usage rules contained in Chapter 5 on page 137. Chapter 5 on page 137 precisely describes the description and formats of data exchanged between application requesters and application servers. See Section 5.3 on page 145 for rules pertaining to this topic.

The following sections define the DRDA rules between an application requester (AR) and an application server (AS). The rules are equivalent between an application server (AS) and a database server (DS) but are not specifically described unless noted in the rule. The terms application requester and application server can be interchanged with application server and database server unless specifically identified in the rule.

7.1 Connection Allocation (CA Rules)

CA1 Only the application requester can initiate network connections between an application requester and an application server.

CA2 Network connections between an application requester and an application server must be started with the required characteristics as defined in the rule usage for the specific network protocol in use (see Part 3, Network Protocols).

See rules usage for environment in these sections:

- Section 12.8.2.1 on page 415
- Section 13.6.2.1 on page 434

CA3 A connection between an application requester and an application server using remote unit of work protocols must not be protected by a sync point manager.

A connection between an application requester and an application server using distributed unit of work can be protected by a sync point manager or be unprotected. If either the application requester or application server does not support a protected connection, the connection must be established without a sync point manager.

See rules usage for environment in these sections:

- Section 12.8.2.1 on page 415
- Section 13.6.2.1 on page 434

CA5 ACCRDB must be rejected with MGRDEPRM when DRDA-required network connection parameters are not specified or are specified incorrectly.

See rules usage for environment in this section:

- Section 12.8.2.1 on page 415

Not applicable in a TCP/IP environment.

CA10 Receivers of ACCRDB must understand the values of TYPDEFNAM and the CCSIDSBC specification of TYPDEFOVR. If the receiver does not understand the values, then it should return VALNSPRM. This should be handled like any other VALNSPRM error on ACCRDB.

Values of CCSIDMBC and CCSIDDBC that the receiver does not understand should be reported with an ACCRDBRM with a WARNING severity. Application requesters can report the warning with SQLSTATE X'01539'. If additional SQL statements use any misunderstood CCSIDs, errors occur. These errors are then reported with an SQLCA indicating data errors along with any reply message that is appropriate to the command that encountered the error.

CA11 Receivers of ACCRDBRM must understand the values of TYPDEFNAM and the CCSIDSBC specification of TYPDEFOVR. If the receiver does not understand the values, then it should terminate the connection. This should be handled like a receipt of a VALNSPRM error in the ACCRDBRM.

Values of CCSIDMBC and CCSIDDBC that the receiver does not understand should be saved for possible problem determination actions later. Application requesters can report the warning with an SQLSTATE of X'01539'. If additional SQL statements use any misunderstood CCSIDs, errors occur. These errors are then reported with an SQLCA indicating data errors along with any reply message that would be appropriate to the command that encountered the error.

When the ACCRDBRM received has the WARNING severity, that fact should be recorded for possible later problem determination actions if it is not reported to the application.

CA12 An application requester using distributed unit of work protocols can initialize a connection with one or more application servers in a unit of work.

CA13 This rule is retired.

CA14 An application requester and application server must provide support for at least one network protocol defined in Part 3, Network Protocols).

7.2 Mapping of RMs to SQLSTATEs (CD Rules)

CD1 If an application requester receives a valid reply message (RM) with a valid *svrcod*, the application requester must return the SQLSTATE listed in Section 8.1 on page 331. If an application requester receives an RM that is not valid in DRDA, or a valid RM with an *svrcod* that is not valid in DRDA, the application requester returns SQLSTATE 58018.

CD2 If an SQLCARD accompanies an RM, the SQLCODE and SQLSTATE in the SQLCARD should be passed to the application.

7.3 Connection Failure (CF Rules)

CF1 When a network connection fails, the application server must implicitly roll back the effects of the unit of work and deallocate all database management resources supporting the application.

CF2 When a network connection fails, the application requester must report the failure to the application in the SQLCA.

7.4 Commit/Rollback Processing (CR Rules)

CR2 Application servers using remote unit of work protocols and application servers using distributed unit of work but not protected by a sync point manager must inform the application requester when the current unit of work at the application server ends as a result of a commit or rollback request by an application or application requester request. This information is returned in the RPYDSS, containing the ENDUOWRM reply message. This RPYDSS is followed by an OBJDSS containing an SQLCARD with information that is input to the SQLCA to be returned to the application. If multiple commit or rollbacks occur prior to exiting a stored procedure, only one ENDUOWRM is returned. See rule CR13 for setting the *uowdsp* parameter when multiple commit and/or rollbacks occur in a stored procedure. See CR6 for the SQLSTATEs to return.

See rules usage for environment in these sections:

- Section 12.8.2.2 on page 416
- Section 13.6.2.2 on page 434

CR3 When a unit of work ends, the application requester must ensure, for all opened cursors that did not have the HOLD option specified, that all query buffers containing unprocessed data (Limited Block Protocols) are purged and that all cursors are in the not open state.

When the HOLD option has been specified for a cursor, a commit does not close that cursor; the application requester must leave that cursor open with its current position in the buffer for the next Fetch.

Note: This includes cursors that were opened with the HOLD option specified within a stored procedure invoked within the unit of work.

CR4 The ending of a network connection causes an application server initiated rollback. The application server assumes termination of the SQL application associated with the connection.

The SQL application should initiate commit or rollback functions prior to termination. If the SQL application terminates normally but does not explicitly commit or rollback, then the application requester must invoke the commit function before terminating the network connection. If the SQL application terminates abnormally, the application requester must invoke the rollback function before terminating the network connection.

CR5 An SQL COMMIT or ROLLBACK, when embedded in the application, is mapped to the DDM commands RDBCMM and RDBRLLBCK, respectively. An SQL COMMIT or ROLLBACK, when executed as dynamic SQL, is mapped to the DDM commands for dynamic SQL—either EXCSQLIMM for EXECUTE_IMMEDIATE or PRPSQLSTT and EXCSQLSTT for PREPARE followed by EXECUTE.

CR6 The parameter *rdbalwupd* of the DDM command ACCRDB is an application requester specification of whether or not the application server is to allow update operations. An update operation is defined as a change to an object at the relational database, such that the change to the object is under commit/rollback control of the unit of work that the application requester initiates.

When the application requester specifies that no updates are allowed, the application server must enforce this specification and, in addition, must not allow the execution of a commit or rollback that the DDM command EXCSQLIMM or EXCSQLSTT requested.

An application requester request that violates the no-update specification is to be rejected with SQLSTATE X‘25000’ for update operations, SQLSTATE X‘2D528’ for dynamic requests to commit, and SQLSTATE X'2D529' for dynamic requests to rollback.

If the local environment allows it, the application requester should initiate processing of commit or rollback for SQLSTATEs X'2D528' and X'2D529'. If the local environment does not allow the application requester to initiate commit or rollback, the SQLSTATEs should be returned to the application.

The application requester may use *rdbalwupd* to ensure that the application performs read-only operations while the application is executing in an environment that supports access to a set of resources such that each member of the set is managed by a distinct resource manager and consistency of the set is controlled by a two-phase commit protocol initiated to the resource managers by the application manager.

CR8 An application server begins commit processing only if it is requested to commit by the sync point manager. If an application requester receives a request to commit from the sync point manager on the connection with an application server, the application requester must ensure a rollback occurs for the unit of work.

See rules usage for environment in this section:

- Section 12.8.2.2 on page 416

Not applicable to TCP/IP.

CR9 An application server protected by a sync point manager can only process dynamic commit or rollback requests or commit requests generated via a stored procedure defined with the *commit on return* attribute if the parm *rdbcmtok* has a value of TRUE indicating the server is allowed to process the commit or rollback.

Otherwise, the application server must refuse the commit or rollback request by returning a CMMRQSRM to the application requester. The *cmmtyp* parameter must indicate the type of request (commit or rollback).

CR10 If an application server is protected by a sync point manager and it receives an RDBCMM or RDBRLLBCK, the RDBCMM or RDBRLLBCK must be rejected and a CMDVLTRM must be returned to the application requester with the *cmmtyp* value identifying the type of request (RDBCMM or RDBRLLBCK).

CR11 If an application server successfully commits through either a EXCSQLSTT or EXCSQLIMM command but a read-only application server with held cursors rolls back, the application requester must inform the application the commit successfully completed. If the next application request is not a static rollback request, the application requester must reject the request and return SQLSTATE 51021 to the application unless the application requester has performed an implicit rollback and informed the application both the commit was successful and an implicit rollback occurred.

In the above situation the server performing the commit could be either a remote unit of work server that is allowed updates or a distributed unit of work server that is allowed to commit via the *rdbcmtok* parameter.

CR12 An application server using distributed unit of work must refuse SQL commit and SQL rollback requests that are inside stored procedures. The refusal to perform the commit or rollback is returned to the stored procedure. The stored procedure logic is responsible to provide the appropriate results to the application.

CR13 The application server must return the results of the rollback in the *uowdsp* on ENDUOWRM if both a rollback and a commit occur inside a stored procedure.

CR14 An application requester cannot send an *rdbcmtok* parameter set to the value TRUE to an application server if that server is connected by a sync point manager, if that server is read only, or if there is another server with uncommitted updates involved in the transaction.

If an application server protected by a sync point manager receives *rdbcmtok* set to the value TRUE the application server should generate an alert and return CMDVLTRM to the application requester.

If a read-only application server receives *rdbcmtok* set to TRUE on a command and a commit or rollback request occurs during execution of the command, then the commit or rollback request should be rejected and an SQLSTATE X‘2D528’ for commit or X‘2D529’ for rollback should be returned to the application requester.

7.5 Connection Usage (CU Rules)

See the DDM Reference for descriptions of the DDM commands.

CU2 The first command required to flow over a DRDA connection is the DDM EXCSAT command.

CU3 The first DRDA command required to flow over a DRDA connection is the DDM ACCRDB command.

CU4 If the application server desires to terminate DDM command chaining, and there is no appropriate DDM RM associated with the SQLCA, the application server must return SQLERRRM to break the chain, if SQLERRRM is a valid reply to the command (for instance, SQLERRRM is not a valid response to BNDSQLSTT).

CU5 *Continue on error* must not be specified in the Data Stream Structure (DSS) header. If specified, a SYNTAXRM with a *synerrcd* =X'04' should be returned.

CU10 The DRDA level selected for use between an application requester and an application server can be no higher than the highest common support level of the two participants. This does not restrict an application requester from operating at different levels to different application servers in the same unit of work.

CU11 An application server that supports the CCSID manager must return a required CCSID manager-level value if the CCSID value received on EXCSAT is one of the required CCSID manager-level values. The required CCSID manager-level values are 500, 819, and 850.

The CCSID manager is not supported using SQLAM Level 3 protocols.

CU12 If a DRDA connection is supported by a SECMGR at Level 5, the initializing EXCSAT must be immediately followed by one and only one ACCSEC / SECCHK exchange. Any other attempts to send ACCSEC or SECCHK when SECMGR is Level 5 should be rejected with PRCCNVRM with *prccnvcd* set to X'10'.

CU13 If commands are chained, then any command which returns EXTDTA reply objects must be the last command in the chain with the same correlation ID. If another command with the same correlation ID is chained after that command, the application server rejects the command with PRCCNVRM with *prccnvcd* set to X'13'.

7.6 Conversion of Data Types (DC Rules)

DC2 Conversion between a DRDA data stream data type and an application variable data type is the responsibility of the application requester.

When converting floating point numbers, use the default rounding rule. That is, round to the nearest value and away from zero in the case of two nearest values.

Exceptions may occur when converting from DRDA data stream data types to application variable data types. The application program receives an SQLSTATE of X‘22001’ for this error.

DC3 To promote interoperability among partners at different SQLAM levels, data types that are supported starting at a given minimum SQLAM level will be subject to data conversion.

If the application requester is at the minimum SQLAM level or higher, then the data description and the data itself are converted before being sent to an application server at a lower SQLAM level as follows:

1. Map any SQL host variable description X in an SQLVRBGRP to that of SQL type Y before sending the SQLVRBGRP.
2. Map any data description X in an SQLDTAGRP to an equivalent DRDA type for Y and convert its corresponding FD:OCA data from its source representation to its equivalent SQL representation as type Y data before sending it.

Source Type (X)	**Mapped Type (Y)**	**Minimum SQLAM Level**
8-byte integer	decimal(19,0)	6
row identifier	varchar(40) for BIT data	6
datalink - SBCS	long varchar(n) for SBCS data	6
datalink - MBCS	long varchar(n) for MIXED data	6
BLOB	Not defined.	6
CLOB - SBCS	Not defined.	6
CLOB - MBCS	Not defined.	6
DBCLOB	Not defined.	6
BLOB locator	Not defined.	6
CLOB locator	Not defined.	6
DBCLOB locator	Not defined.	6

If no data conversion is defined, the behavior depends on the descriptor group. If the descriptor group is the SQLVRBGRP, the source SQL descriptor is sent to the application server. If the descriptor group is the SQLDTAGRP, the application requester rejects the command with an SQLSTATE of 56084.

DC4 To promote interoperability among partners at different SQLAM levels, data types that are supported starting at a given minimum SQLAM level will be subject to data conversion.

If the application server is at the minimum SQLAM level or higher, then the data description and the data itself are converted before being sent to an application requester at a lower SQLAM level as follows:

1. Map any SQL host variable description X in an SQLDAGRP to that of SQL type Y before sending the SQLDAGRP.

2. Map any data description X in an SQLDTAGRP to an equivalent DRDA type for Y and convert its corresponding FD:OCA data from its source representation to its equivalent SQL representation as type Y data before sending it.

Source Type (X)	Mapped Type (Y)	Minimum SQLAM Level
8-byte integer	decimal(19,0)	6
row identifier	varchar(40) for BIT data	6
datalink - SBCS	long varchar(n) for SBCS data	6
datalink - MBCS	long varchar(n) for MIXED data	6
BLOB	Not defined.	6
CLOB - SBCS	Not defined.	6
CLOB - MBCS	Not defined.	6
DBCLOB	Not defined.	6
BLOB locator	Not defined.	6
CLOB locator	Not defined.	6
DBCLOB locator	Not defined.	6

If no data conversion is defined, the behavior depends on the descriptor group. If the descriptor group is the SQLDAGRP, the source SQL descriptor is sent to the application requester. If the descriptor group is the SQLDTAGRP, the application server rejects the command with an SQLSTATE of 56084.

DC5 To promote consistent behavior among DRDA partners who choose to provide differing levels of support for DRDA types, data types that are supported starting at a minimum SQLAM level will be subject to data conversion.

If an application requester at a given SQLAM level does not support a given data type defined at that SQLAM level, then before presenting the data or its description to the application, it must use the mapping defined in rule DC3 to convert the data or descriptor received from an application server that does support that data type. If no mapping is defined, the application requester rejects the command with an SQLSTATE of 56084.

If an application server at a given SQLAM level does not support a given data type defined at that SQLAM level, then before presenting the data or its description to the relational database, it must use the mapping defined in rule DC4 to convert the data or descriptor received from an application requester that does support that data type. If no mapping is defined, the application server rejects the command with an SQLSTATE of 56084.

7.7 Data Representation Transformation (DT Rules)

DT2 The data representation for all DRDA command input and output data other than ACCRDB is in the format defined by the TYPDEFNAM and overrides (TYPDEFOVR) exchanged on ACCRDB and ACCRDBRM or included in command or reply data objects to override specifications for a particular command or object.

Note: DDM command parameters and reply message parameters are *not* considered as input and output data. DDM defines representation of these parameters. Only command data objects and reply data objects are affected by the TYPDEFNAM that the ACCRDB command specified. Refer to Section 4.4.1 on page 54 for more details on the ACCRDB DDM command.

DT3 All data representation transformations are the responsibility of the receiver of the data object. With the exception of character data types, application servers do data representation transformation for data received from application requesters; application requesters do data representation transformation for data received from application servers.

For all character data types that are received from the application requester (such as data types that carry CCSIDs) the relational database has the responsibility of data representation transformation when necessary.

For all data types that are received from the application server, the SQLAM has the responsibility of data representation transformation when necessary. The DRDA Reference defines all conversions of character data between CCSIDs.

DT4 A data representation transformation error (no representation of the character in the application server CCSID) may occur when the application server transforms application input string variable values, which the application server received from the application requester, to its representation. The application program receives an SQLSTATE of 22021 for this error.

DT5 A data representation transformation error (no representation of the character in the application requester code page) may occur when the application requester transforms string values, which the application requester received from the application server, to its representation.

If the string value cannot be assigned to an application variable that has an indicator variable, then the application program receives a warning SQLSTATE of 01520. If the string value cannot be assigned to an application variable that does not have an indicator variable, then the application program receives an error SQLSTATE of 22021.

DT6 An overflow error may occur when the application requester transforms a floating point number, which the application requester received from the application server, to its representation.

If an application variable size mismatch occurs for a value being returned to the application program and the application variable has an indicator variable, then the application program receives a warning SQLSTATE of 01515.

If an arithmetic exception occurs for a value being returned to the application program and the application variable has an indicator variable, then the application program receives a warning SQLSTATE of 01519.

If an application variable size mismatch occurs for a value being returned to an application program and the application variable does not have an indicator variable, then the application program receives an error SQLSTATE of 22001.

If an arithmetic exception occurs for a value being returned within an inner SELECT or for a value being returned to an application variable in an application program that does not have an indicator variable, then the application program receives an error SQLSTATE of 22003, 22012, 22502, or 22504.

DT7 When transforming a floating point number (such as 370 floating point to IEEE floating point), round to the nearest value and away from zero in the case of two nearest values.

DT8 If the representation of the data to be sent is different than the representations agreed to at ACCRDB, then the application requester or the application server adds TYPDEFNAM and TYPDEFOVR parameters to command or reply data objects, as necessary, to correctly describe the data being sent. FDODSC and QRYDSC objects do not change.

For a given command, the TYPDEFNAM and TYPDEFOVR objects are sent as command data objects. The data representations of all the following command data objects of that command are affected. The early and late group, row, and array descriptors for these command data objects take their representations from these TYPDEFNAM and TYPDEFOVR values. The same rules apply when TYPDEFNAM and TYPDEFOVR objects precede any reply data objects returned to the command.

DT9 TYPDEFNAM may be specified as many times as necessary to correctly describe all objects required for a command or returned in the reply to a command.

The overrides are in effect for only one command or the reply to one command.

DT10 The representation for all data received in QRYDTA objects from a single query (SQLCAs and user data, including EXTDTA objects) is determined by ACCRDB/ACCRDBRM or overrides effective at the time the QRYDSC is received. If the application requester sends an OUTOVR object with a CNTQRY command, the TYPEDEFNAM, TYPDEFOVR associated with the QRYDSC applies to the OUTOVR object as well.

An SQLDARD is intended to be converted to an SQLDA for the application program and should not be used as a description of the data on the wire. If the application requester has received an SQLDARD for this section, then the description contained in the SQLDA is returned to the application. The application requester does not use the SQLDA as the basis for determining the representation of the data sent from the application server. The sole determinant of data representation is the QRYDSC with the TYPDEFNAM, TYPDEFOVR, specified on ACCRDBRM or any override received prior to the QRYDSC object.

DT11 If an application requester cannot process a new value for TYPDEFNAM that is received from an application server as part of a reply data object, then it must produce an SQLCA for the application. The SQLCA indicates SQLSTATE 58017, specifying the parameter that the application server requested, but that the application requester could not support.

DT12 If an application server cannot process the new values for TYPDEFNAM that it received from an application requester as part of a command data object, then it must return VALNSPRM to the application requester. The application requester will handle this like any other VALNSPRM error.

DT13 If an application requester cannot process data according to the CCSID specified for this data, then it must produce an SQLCA for the application indicating SQLSTATE 57017 specifying the pair of CCSIDs for which conversion could not be performed.

DT14 If an application server cannot process data according to the CCSID specification for this data then it must return an SQLCA indicating SQLSTATE 57017 specifying the pair of CCSIDs for which conversion could not be performed.

DT15 The CCSID specified on a TYPDEFOVR overrides only the corresponding CCSID type on the ACCRDB/ACCRDBRM for the duration of the command or reply, and only until the corresponding CCSID type in the next TYPDEFOVR is found on the command or reply. At completion of the command or reply, all CCSID specifications revert to those established by ACCRDB or ACCRDBRM.

DT16 If the sender has not specified CCSIDMBC or CCSIDDBC on an ACCRDB/ACCRDBRM, nor on a TYPDEFOVR of a command/reply data object, then character data of that representation should not be sent unless explicitly defined by MDD/SDA pairs.

The receiver of this data should return an SQLCA indicating SQLSTATE 57017 with zero as the source CCSID token.

DT17 An application requester must change all non-nullable data types for host variables associated with a statement that invokes a stored procedure (that is, CALL statement) to the nullable version of the data type before sending the request to the application server.

DT18 An application server must set the indicator variables for INPUT host variables associated with a statement that invokes a stored procedure (that is, CALL statement) to −128 prior to returning the host variables to the application requester.

DT19 TYPDEFNAM or TYPDEFOVR objects are ignored for any EXTDTA blocks or for extra query blocks. For any given EXTDTA object, the overrides in effect for the object containing the associated FD:OCA placeholder are also in effect for the EXTDTA.

DT20 A DATALINK data column may exist in a database management system and be presented to an application as a structure containing non-character (viz, binary) data. However, when a DATALINK column flows on the wire it must conform to the following format. The two-byte length prefix must be set to the length of the string that follows, as with a LONG VARCHAR type, and the contents of the string be as shown below:

Position	Field Name	Description
1-5	VERSION	Character form of INTEGER version number, padded with leading zeros.
6-9	LINK_TYPE	Four-byte character string indicating the link type.
10-14	URL_LENGTH	Character form of INTEGER length of the following URL field, padded with leading zeros (assumes a length ≤ 99999).
15-22	(reserved)	Eight blanks in initial version.
23-xx	URL	Character string containing the URL of the associated file, whose ending position, xx, is URL_LENGTH + 22.
yy-zz	COMMENT	Character string containing a comment about the DATALINK, whose starting and ending positions are URL_LENGTH + 23 (yy) and the value of the 2-byte string prefix (zz).

Note: The implementer of a DRDA application requester has complete freedom of choice as to what to do with a received string for a DATALINK column. One reasonable option is to extract the URL portion and return that to the user. An application requester may choose to include the comment with the URL. The DRDA architecture does not specify how the string is used. Normally, an application would use a scaler function on the column to extract the desired potion of the complete structure, in which case the DRDA type would be that of the function and not be the DATALINK type.

7.8 RDB-Initiated Rollback (IR Rules)

IR1 If the local environment at the application server does not initiate a global rollback when the application server detects a relational database-initiated rollback, it must send the reply message ABNUOWRM (RDB-initiated rollback) as the response to the application requester request. Normally, this is the request currently being serviced.

However, if the request is an OPNQRY or CNTQRY using the Limited Block Protocol, the response may have to be deferred until the next CNTQRY, adhering to rules QT2 through QT5 (see Section 7.19.3 on page 318). If the application server defers the ABNUOWRM reply message, it must return the message as the response to the next application requester command request, regardless of the type of request.

IR2 If an application requester receives a request to back out from the network facility on a network connection with an application server, the application requester must ensure that rollback occurs at all application servers involved in the unit of work.

7.9 Optionality (OC Rules)

OC2 Application requesters do not have to send optional commands or optional parameters on any command or all possible values of any parameter unless explicitly stated in this volume.

OC3 Application servers must recognize all optional commands, parameters, and values. They are allowed to reject optional commands and any commands that have optional parameters that contain values other than the default for the optional components. The application server might also reject required parameters that contain values not defined by DDM as permissible values or that have lengths within the permissible range supported by DDM but beyond the maximum length supported by the application server. This should be reported with one of the four DDM *not supported* reply messages. These are:

- CMDNSPRM for unsupported commands
- PRMNSPRM for unsupported parameters
- VALNSPRM for unsupported values
- OBJNSPRM for unsupported objects

OC4 Application servers do not have to send optional parameters of reply messages or reply data objects. Application servers do not have to send every possible reply message. In fact, the circumstances at a particular application server might make it impossible to get to the situation an RM covered.

OC5 Application requesters must recognize all optional parameters and values sent in reply messages and reply data objects. They are allowed to discard any optional information unless explicitly stated otherwise in this volume.

OC6 Application requesters must be prepared to receive *Not Supported* reply messages for any optional components they send to an application server.

OC8 When an application requester does not specify an optional parameter that the target application server supported, the application server must apply the default rules specified in DDM.

OC9 When the end user and/or the application does not supply a parameter value, and the parameter is required, the application requester must return an error message or apply an application requester value to the parameter and specify the parameter on the command.

OC10 When the end user and/or the application does not supply a parameter value, and the parameter is optional, the application requester must not include the parameter on the command but must allow the application server to apply the default value.

7.10 Program Binding (PB Rules)

PB1 The relational database name (RDB_NAME) contained in the package name supplied on the BGNBND command must be the same as the RDB_NAME supplied on the ACCRDB command.

PB2 After the application requester sends a BGNBND command to the application server and receives a non-error response, the only valid command request to this application server before ENDBND, RDBCMM, RDBRLLBCK, or resource recovery processing is BNDSQLSTT.

PB3 The BNDSQLSTT command is valid only between the BGNBND and resource recovery processing or between BGNBND and one of these commands: ENDBND, RDBCMM, or RDBRLLBCK.

PB4 After the application requester sends a BGNBND command to the application server and receives a non-error response, the package name supplied on the BNDSQLSTT and ENDBND commands must be the same as the package name supplied on the BGNBND command.

PB5 A new package that DRDA bind command sequence has bound becomes persistent only after a commit.

PB6 If a rollback occurs prior to a commit, a DRDA bind command sequence does not replace an old package.

PB7 A commit performs an implicit ENDBND.

PB8 A package can be dropped and then recreated without an intervening commit. Conversely, a package can be created and then dropped without an intervening commit.

PB9 SQL statements in an application program are input to the BIND process by a BNDSQLSTT. The following statements are exceptions and should not flow at bind: INCLUDE, WHENEVER, PREPARE, EXECUTE, EXECUTE IMMEDIATE, DESCRIBE, OPEN, FETCH,[41] CLOSE, COMMIT, CONNECT, ROLLBACK, RELEASE, SET CONNECTION, DISCONNECT, BEGIN DECLARE SECTION, END DECLARE SECTION, and local statements.[42]

The SQL statement is the SQLSTT command data object of the BNDSQLSTT command.

The processing for an individual SQL statement that the target relational database performs is specific to the environment.

PB11 The order in which SQL statements must be submitted to the relational database's BIND process is defined in *ISO/IEC 9075: 1992, Database Language SQL.* For example, the declaration of a SELECT must precede the corresponding OPEN, FETCH, and CLOSE.

PB12 Each application variable referenced in an SQL statement to be bound must be described by an SQLDTA FD:OCA description in the order in which the application

41. A connection using distributed unit of work protocols, the FETCH statement can flow to distributed application servers during the bind process. (See rule Section 7.10).

42. A local statement is understood by the precompiler and either processed completely by the precompiler, or it results in a call to the application requester at runtime, which does not cause any flows to the application server.

variable appears in the SQL statement.

This includes a program variable reference that specifies a procedure name within an SQL statement that invokes a stored procedure. Note, however, that the stored procedure name value flows in the *prcnam* parameter rather than in an SQLDTA on the EXCSQLSTT for that SQL statement. The set of application variables so described is the SQLSTTVRB command data object of the BNDSQLSTT command.

PB13 Any application variable references that show indicator variable usage map to a pair of variables. The first variable has the characteristics of the user's true data. The second variable is a SMALL INTEGER and represents the indicator variable.

When the user's data is sent at execution time, it is just one variable. That variable is nullable if the corresponding column is nullable.

PB14 If an application server, or the relational database associated with the application server, does not include in its BIND process a particular SQL statement, the response to the application requester for such an SQL statement is an SQLCARD reply data object with an error SQLSTATE.

PB17 The character string *:H* replaces each application variable reference (user data or indicator) before it is sent to the application server for BIND.

It is allowable to have one or more blanks between the : and H.

PB19 SQL statements that the application requester does not understand are sent with the following assumptions:

- All host variables are input variables.
- The statement is assigned a unique section number.
- The section is executed by an EXCSQLSTT command.

The BNDSQLSTT *bndsttasm* parameter is used to alert the application server of these assumptions. If the assumptions are incorrect, the application server returns an SQLSTATE of X'42932' for that statement. The application server has the final word on validity of the statement.

A statement is not understood when the application requester cannot classify the statement properly. That is, the application requester does not know the statement type, or the application requester cannot tell which host variables are input or output.

PB20 If the application requester language processor supports structure or array references to provide shorthand notation to refer to many program variable fields, then *:H*s are inserted into the SQL statement for each element of the structure or array. Commas separate these *:H*s (for example, :H,:H,:H for a three element structure or array).

If there is an indicator structure specified in the program variable reference, and if the data structure has *m* more variables than the indicator structure, then the last *m* variables of the data structures do not have the indicator variables.

If the data structure has *m* less variables than the indicator structure, the last *m* variables of the indicator structure are ignored. Each substitution, if there is an indicator variable, then becomes a pair of *:H*s (for example, :H:H,:H:H,:H for a data structure with 3 variables, and an indicator array with 2 elements).

It is allowable to have one or more blanks between the : and H.

PB26 A single variable represents any application variable references that do not show indicator variable usage. That variable must use the non-nullable data type. See rule

PB13 for nullable cases.

PB27 If the application server receives an ENDBND to terminate bind processing, and an error occurred during bind processing that prevents the successful generation of the package, the SQLSTATE in the SQLCARD that the application server generates must not begin with the characters 00, 01, or 02. The values 00, 01, and 02 imply the package was created. All other values imply the package was not created.

PB28 An application requester that supports multi-row and/or scrollable cursors must flow a FETCH statement at bind time if the FETCH statement contains multi-row and/or scroll parameters. An application server that does not support multi-row and/or scrollable cursors must reject the FETCH statement at bind time. If the application server supports multi-row and/or scrollable cursors, it must accept the FETCH at bind time. If the application server accepts FETCH statements at bind time, the application server will use the section number associated with the cursor.

This rule does not apply to remote unit of work components.

PB29 BNDOPT should not be used to flow bind options and values for which codepoints are explicitly defined in DRDA. For example, do not use BNDOPT to send the option ISOLATION_ LEVEL = CURSOR_STABILITY to a server since PKGISOLVL has been created for this purpose. Conflicts in bind options are detected by the application server and are reported by returning an SQLSTATE of X‘56096’ to the application requester.

7.11 Security (SE Rules)

SE2 The application server must be able to obtain the verified end user name associated with the connection.

See rules usage for environment in these sections:

- Section 12.8.2.3 on page 416
- Section 13.6.2.3 on page 435

SE3 If user identification and authentication security is not provided using SECMGR Level 5, an application requester must have send support for the types of security defined for the specific network protocols defined in Part 3, Network Protocols. An application server must have receive support for the types of security defined for the specific network protocols defined in Part 3, Network Protocols. For example, if an end-user name is provided on a network connection, the end-user name supplied in the DCE security token takes precedence over the end-user name received from the network facility.

See rules usage for environment in this section:

- Section 12.8.2.3 on page 416

SE5 If SECMGR is at Level 5, the application requester and application server must support at least one of the security mechanisms defined in Chapter 10 on page 343.

SE6 Connections using the DCE security mechanism do not use GPSS channel bindings.

7.12 SQL Section Number Assignment (SN Rules)

SN1 A section number is between 1 and 32,767 inclusive.

SN2 When a statement requires the assignment of a unique section number, a section number one larger than the previous number allocated is assigned. If this is the first statement to be assigned a number, then it is assigned section number 1.

During the bind process, the application server can receive section numbers out of sequence. The same section number is assigned to related SQL statements (see rule SN3), but not all of these statements are sent to the application server during bind processing (see rule Section 7.10 on page 296). Therefore, the first occurrence of a section number the application server receives might not be the first SQL statement in the related group. Unrelated statements can be interspersed among related statements that share a section number.

An application server can, but is not required to, allow SQL statements that are not part of a related statement group to arrive out of sequence.

At the conclusion of bind processing, gaps in the section numbers can exist in the package. These gaps are the result of dynamic SQL statements that were not sent during the bind process (see rule Section 7.10 on page 296), but may be referenced at execution time.

SN3 The application requester assigns the same section number to all related SQL statements that have execution time dependencies. Specifically, the application requester assigns each declared statement or cursor a unique section number. A cursor declared for a statement shares the statement section number.

Each SQL statement that references the declared statement or cursor (FETCH, EXECUTE, OPEN, CLOSE, PREPARE) receives the same section number as the referenced statement or cursor.

SN4 The application requester assigns a unique section number to the statements ALTER, CALL, COMMENT ON, CREATE, DELETE, DROP, EXPLAIN, GRANT, INSERT, LABEL ON, LOCK, SELECT (embedded), REVOKE, and UPDATE.

SN5 Each occurrence of EXECUTE_IMMEDIATE may be assigned a unique section number, share a section number of one or more other EXECUTE_IMMEDIATEs, or all EXECUTE_IMMEDIATEs may share the same section number.

SN7 The largest section number the application requester assigns to any statement is communicated to the application server by the *maxsctnbr* parameter on the DDM command ENDBND. Any gap between the highest section seen and the value of *maxsctnbr* will be available for sections with dynamic statements.

SN8 Each section number that the application requester sends to the application server must be unique. (For the related statements OPEN, FETCH, CLOSE and DECLARE CURSOR, the DECLARE CURSOR is sent and FETCH is conditionally sent. See rule Section 7.10 on page 296). The application server can process or discard any statement with a duplicate section number that is subsequently received.

SN9 A section number may be repeated in flows to the application server when the immediately prior statement was bound with errors and the current statement's section number matches that on the prior statement. In this case, the same section number may be sent again.

The result of subsequent binds of the same section number reset error indicators previously set for that section number.

7.13 Stored Procedures (SP Rules)

SP1 If both *pkgnamcsn* and *prcnam* are specified on an EXCSQLSTT for a CALL or other SQL statement that invokes a stored procedure, then:

- If the section identified by *pkgnamcsn* exists in the package identified by *pkgnamcsn*, but the section is not associated with a stored procedure, then the use of *prcnam* with *pkgnamcsn* is invalid and the application server returns CMDCHKRM to the application requester.
- If the CALL or other SQL statement specifies the procedure name using a host variable, the section identified by *pkgnamcsn* exists in the package identified by *pkgnamcsn*, and the section is associated with a stored procedure, then the application server invokes the stored procedure by using the *prcnam* value.
- If the CALL or other statement that invokes a stored procedure does not specify the procedure name using a host variable, then the value specified by the *prcnam* parameter, if present, must match the procedure name value contained within the section identified by *pkgnamcsn*.

SP2 If there are any host variables in the parameter list of a stored procedure (that is, CALL statement), the presence of all variables should be reflected (by a null indication or data) in both the SQLDTA that flows from the application requester, and the SQLDTARD that is returned from the application server.

SP3 If a CALL or other statement that invokes a stored procedure specifies the procedure name using a host variable, then the *prcnam* parameter of the EXCSQLSTT specifies the procedure name value. The procedure name value is not duplicated in any SQLDTA command data object that might also flow with the EXCSQLSTT.

SP4 In situations where a single application requester connects to more that one application server during the execution of a client application, the application requester may receive the same locator value within the SQLRSLRD from more than one application server. It is the responsibility of the application requester to ensure that a locator value returned to a client application is unique for a particular execution of that client application.

7.14 SET Statement (ST Rules)

ST1 Non-local SET statements that set the content of a special register flows at BIND. The following are local SET statements:

- SET CONNECTION
- SET CURRENT PACKAGESET

All other or unrecognized SET statements are considered non-local.

ST2 An application requester does not automatically propagate the setting of special registers at the current application server when the application requester connects to a new application server.

An application requester that flows non-local SET statements need *not* track the effect of SET statements that set the contents of special registers at an application server.

Any application server that connects to a database server must track the execution of SET statements and the effect of the contents of special registers. Prior to executing an SQL command at a database server, any new or changed settings must be propagated using the DDM EXCSQLSET command. EXCSQLSET contains an ordered list of SET statements. The SET statements are used to set the special registers to the values at the application server.

ST3 Non-local SET statements should be executed at an application server and a database server in the order received.

ST4 If an application server or database server does not recognize a SET statement, it must return a warning SQLSTATE with an SQLCARD object.

If a database server recognizes the SET statement but the processing of the statement fails that should prevent the processing of any other SQL statements, the database server must return an SQLERRRM reply message with an SQLCARD object.

7.15 Serviceability (SV Rules)

SV1 The application requester must generate diagnostic information and may notify a network focal point when it receives an abnormal disconnect of the network connection from the application server.

See rules usage for environment in these sections:

- Section 12.8.2.4 on page 417
- Section 13.6.2.4 on page 435

SV2 The application requester must generate diagnostic information and may notify a network focal point when it receives the following DDM reply messages:

- AGNPRMRM svrcods 16,32,64
- CMDCHKRM svrcods 8,16,32,64
- CMDVLTRM svrcod 8
- DSCINVRM svrcod 8
- DTAMCHRM svrcod 8
- PRCCNVRM svrcods 8,16,128
- QRYNOPRM svrcod 8
- QRYPOPRM svrcod 8
- RDBNACRM svrcod 8
- RDBACCRM svrcod 8
- SECCHKRM svrcod 16
- SYNTAXRM svrcod 8

SV3 The application requester must generate diagnostic information and may notify a network focal point when the application requester reaches a resource limit that prevents continued normal processing.

SV4 The application requester must generate diagnostic information and may notify a network focal point when a blocking rule is violated in the data received from the application server.

SV5 The application requester must generate diagnostic information and may notify a network focal point when a chaining rule is violated in the data received from the application server.

SV6 The application server must generate diagnostic information and may notify a network focal point when it generates the following DDM reply messages:

- AGNPRMRM svrcods 16,32,64
- CMDCHKRM svrcods 8,16,32,64
- CMDVLTRM svrcod 8
- DSCINVRM svrcod 8
- DTAMCHRM svrcod 8
- PRCCNVRM svrcods 8,16,128

- QRYNOPRM svrcod 8
- QRYPOPRM svrcod 8
- RSCLMTRM svrcods 8,16,32,64,128
- RDBNACRM svrcod 8
- RDBACCRM svrcod 8
- SECCHKRM svrcod 16
- SYNTAXRM svrcod 8

SV8 The unit of work identifier must be present in the network focal point message, in the supporting data information, and in diagnostic information.

See rules usage for environment in these sections:

- Section 12.8.2.4 on page 417
- Section 13.6.2.4 on page 435

SV9 In a distributed unit of work environment, an application requester must send a correlation token to the application server at ACCRDB using the *crrtkn* parameter. If a correlation token exists for this unit of work, and it has the format the correlation token as defined in Part 2, Environmental Support, then this token is used. If the existing token does not have the correct format, or the token does not exist, then the application requester must generate a correlation token.

See rules usage for environment in this section:

- Section 12.8.2.4 on page 417

SV10 In a distributed unit of work environment, the *crrtkn* value must be present in the network focal point message, in the supporting data information, and in diagnostic information.

7.16 Update Control (UP Rules)

UP1 If the application is not using the services of a sync point manager in the logical unit of work:

- When connecting to an application server using remote unit of work, the application server is only allowed updates if either there are no existing connections to any other application servers, or all existing connections are to application servers using remote unit of work, and these application servers are restricted to read-only.
- If a connection exists to an application server using remote unit of work with update privileges, all other application servers are restricted to read-only. Otherwise, for the duration of any single logical unit of work, the first application server using distributed unit of work that performs an update is given update privileges, and all other application servers are restricted to read-only.

UP2 If the application is using the services of a sync point manager in a unit of work, only connections to application servers using distributed unit of work and protected by a sync point manager are allowed update privileges.

UP3 Within a distributed unit of work, an application server must return an RDBUPDRM the first time a DDM command results in an update at the application server. An application server can, but is not required to, return an RDBUPDRM after subsequent commands in the same logical unit of work that result in an update at the application server.

The sending of RDBUPDRM is not supported when using SQLAM Level 3.

UP4 If there are multiple DDM reply messages in response to a DDM command of which one is an RDBUPDRM, the RDBUPDRM must be the first reply message in the chain of reply messages.

The receipt of RDBUPDRM is not supported when using SQLAM Level 3.

7.17 Passing Warnings to the Application Requester (WN Rules)

WN1 When constructing a response to OPNQRY or EXCSQLSTT that contains answer set data, the application server is responsible for obtaining an SQLDA for the answer set that the relational database will deliver. This data area (DA) specifies:

- The maximum lengths of all variable-length results
- The nullability of any result value
- The derivation of a result value (such as col1/col2 is derived)
- CCSID of a character result value

This data area is used to determine which fields require the application server to provide indicator variables.

WN2 For all variable-length result fields, the application server must provide space to accommodate the maximum length result so that truncation does not occur when the data is delivered from the relational database. This allows the relational database to avoid all truncation warning or error reports.

WN3 For all nullable and derived fields, an indicator variable must be provided so that the null conditions can be reported and errors can be avoided. For derived result values (such as col1/col2), an indicator variable must be provided to allow the relational database to report problems as warnings instead of errors.

WN4 The FD:OCA descriptor for all nullable and derived fields must use an FD:OCA nullable data type.

WN5 The application requester is responsible for taking null indicators from FD:OCA data (1 leading byte) and converting them to values for indicator values. The following cases can occur:

- Null indicator 0 to 127 (positive); a data value will follow. The data should be placed in the host value. If truncation occurs, handle as SQL describes and fill in any indicator variable the application provides.
- Null indicator –1 to –128 (negative); no data value will follow.
  - — If indicator variable is available, fill it with the value from the null indicator.
  - — If indicator variable is unavailable, turn SQL warning code into corresponding error code. The application requester may also need to issue CLSQRY to the application server that issues a close query to the relational database in order to enforce the SQL semantics that the cursor is unusable after the error.

7.18 Names

The following sections define the rules for end-user names, SQL object names, relational database names, and target program names.

7.18.1 End-User Names (EUN Rules)

EUN1 Character strings that represent end-user names or components of end-user names within DRDA flows must contain only Character Set 1134 (uppercase A through Z and 0 through 9).

7.18.2 SQL Object Names (ON Rules)

ON1 DRDA requires that an application server support the receipt of three-part names for tables, views, and packages. The following rules summarize the DRDA three-part naming convention for tables, views, and packages (refer to Chapter 6 on page 269 for a detailed description of the syntax and semantics of three-part names).

ON1A The globally unique fully qualified name for a table or view is RDB_NAME.COLLECTION.OBJECTID. The maximum length of COLLECTION is 18 bytes. The maximum length of an OBJECTID is 18 bytes. COLLECTION and OBJECTID have the same syntactic constraints as SQL identifiers but are limited to SBCS CCSIDs.

ON1B The fully qualified name for a package (database management system access module) is RDB_NAME.COLLECTION.PACKAGEID. The maximum length of COLLECTION is 18 bytes. The maximum length of a PACKAGEID is 18 bytes. The COLLECTION and PACKAGEID have the same syntactic constraints as SQL identifiers but are limited to SBCS CCSIDs.

The period is the delimiter for components of a package name.

ON1C The fully qualified name for a section is PACKAGENAME.SECTION_NUMBER. The maximum length of a SECTION_NUMBER is 2 bytes. A section number is a 2 byte non-negative binary integer.

ON1D The fully qualified name for a stored procedure is RDB_NAME.COLLECTION.PROCEDURE. The maximum length of COLLECTION is 18 bytes. The maximum length of a PROCEDURE is 18 bytes. The COLLECTION and PROCEDURE have the same syntactic constraints as SQL identifiers but are limited to SBCS CCSIDs.

The period is the delimiter for components of a stored procedure.

7.18.3 Relational Database Names (RN Rules)

RN1 An RDB_NAME has the same syntactic constraints as SQL identifiers with the exception that RDB_NAME cannot contain the alphabetic extenders for national languages (#, @, and $, for example). The valid characters are uppercase letters (A through Z), the numerics (0 through 9), and the underscore character (_).

The maximum length of an RDB_NAME is 18 bytes.

RN2 DRDA associates an RDB_NAME with a specific program at a unique network location. DRDA, however, does not define the mechanism that derives the program and network location from the RDB_NAME. The particular derivation mechanisms are specific to the environment.

It is the responsibility of the application requester to determine the RDB_NAME name of the relational database and to map this name to a program and network location.

See rules usage for environment in these sections:

- **LU 6.2 Usage of Relational Database Names Rules** on page 418
- Section 13.6.2.5 on page 435

RN3 More than one RDB_NAME may exist for a single network location.

See rules usage for environment in these sections:

- **LU 6.2 Usage of Relational Database Names Rules** on page 418
- Section 13.6.2.5 on page 435

RN4 DRDA permits the association of more than one RDB_NAME with a single program at a network location.

See rules usage for environment in these sections:

- **LU 6.2 Usage of Relational Database Names Rules** on page 418
- Section 13.6.2.5 on page 435

7.18.4 Target Program Names (TPN Rules)

TPN1 The program names identifying implemented DRDA application servers can be a registered DRDA program name, a registered DDM program name, or any non-registered program name.

See rules usage for environment in these sections:

- **LU 6.2 Usage of Transaction Program Names Rules** on page 418
- Section 13.6.2.6 on page 435

TPN2 DRDA allows DDM file servers and DRDA SQL servers to use either the same program name or different program names.

See rules usage for environment in these sections:

- **LU 6.2 Usage of Transaction Program Names Rules** on page 418
- Section 13.6.2.6 on page 435

TPN3 Registered DRDA program name structures for the specific network protocols are defined in Part 3, Network Protocols.

See rules usage for environment in these sections:

- **LU 6.2 Usage of Transaction Program Names Rules** on page 418
- Section 13.6.2.6 on page 435

TPN4 Multiple DRDA program names may exist for a single network location.

See rules usage for environment in these sections:

- **LU 6.2 Usage of Transaction Program Names Rules** on page 418
- Section 13.6.2.6 on page 435

TPN5 A DRDA program name is unique within a network location.

See rules usage for environment in these sections:

- **LU 6.2 Usage of Transaction Program Names Rules** on page 418
- Section 13.6.2.6 on page 435

TPN6 Target programs that are registered DRDA program names must provide all the capabilities that DRDA requires.

See rules usage for environment in these sections:

- **LU 6.2 Usage of Transaction Program Names Rules** on page 418
- Section 13.6.2.6 on page 435

TPN7 Target programs that provide DRDA capabilities may perform additional non-DRDA work. These target programs are not required to perform additional non-DRDA work.

See rules usage for environment in these sections:

- **LU 6.2 Usage of Transaction Program Names Rules** on page 418
- Section 13.6.2.6 on page 435

TPN8 The registered default DRDA program names for the specific network protocols are defined in Part 3, Network Protocols. The default DRDA program name must be definable at each system that supports at least one application server providing DRDA capabilities.

See rules usage for environment in these sections:

- **LU 6.2 Usage of Transaction Program Names Rules** on page 418
- Section 13.6.2.6 on page 435

7.19 Query Processing

For query processing, there are rules for blocking, query data transfer protocols, and terminating, interrupting, and continuing query data or result set transfer.

7.19.1 Blocking

Blocking refers to the process of sending query reply messages and query reply data objects in uniformly sized units, known as *query blocks*. The application requester specifies the size of each such unit. The application server strives to return all query responses in units of the desired size by combining reply messages and reply data objects smaller than the specified block size into a single query block or by breaking objects larger than the specified block size into multiple query blocks.

All query reply messages and reply data objects are subject to blocking, except for EXTDTA objects. Instead, EXTDTA objects are sent in query blocks that contain only the EXTDTA and are as large as needed to contain the entire externalized FD:OCA data item.

In the blocking rules below, the following terminology is used: All query responses are returned in units called query blocks. Query blocks that must adhere to the block size specification are known as *standard query blocks*. Query blocks that are exempt from the block size specification are known as *non-standard query blocks*.

The blocking rules are based on a minimum block size of 512 bytes for standard query blocks (see rule BS2 in Section 7.19.1.2 on page 313). This size should be kept in mind when reading the rules. To get a good understanding of any one rule, other blocking rules must be understood. Apparent errors and misunderstandings in some rules may be resolved when read in conjunction with other rules.

7.19.1.1 Block Formats (BF Rules)

Given that each DDM Data Stream Structure (DSS) is one SNA logical record, DRDA defines a query data or result set transfer block to consist of one or more SNA logical records such that:

BF1 In a response to an OPNQRY or a CNTQRY command, each block sent, except the first and last block of the answer set, begins with the DDM OBJDSS structure.

- The first block begins with an RPYDSS. See rule QP2 in Section 7.19.2 on page 316.
- The last block may begin with either an RPYDSS or an OBJDSS. See rules QT1, QT2, QT3, and QT4 in Section 7.19.3 on page 318.

BF2 In a response to an EXCSQLSTT command, each block sent begins with either the DDM OBJDSS or the DDM RPYDSS structure.

- The first block begins with an RPYDSS. See rule QP2 in Section 7.19.2 on page 316.
- An intermediate block begins with either an OPNQRYRM RPYDSS or an OBJDSS. See rule BF9, rules BF14, BF15, BF16, and BF17, and rule QP2 in Section 7.19.2 on page 316.
- The last block may begin with either an RPYDSS or an OBJDSS. See rules QT1, QT2, QT3, and QT4 in Section 7.19.3 on page 318.

BF3 The FD:OCA description of answer set data for a query is contained in an OBJDSS that is chained from an OPNQRYRM RPYDSS structure (see chaining rule CH1 in Section 7.19.1.3 on page 314).

The FD:OCA description of the answer set data may be totally contained in the block containing the OPNQRYRM or may overflow to subsequent blocks. If overflow occurs, a two-byte length field and the DDM code point (QRYDSC) of the FD:OCA description must follow the OBJDSS structure in each overflow block. The next byte of the FD:OCA description follows the QRYDSC code point.

The FD:OCA description of answer set data for a result set is contained in an OBJDSS that is chained from an SQLCINRD OBJDSS structure (see chaining rule CH1 in Section 7.19.1.3 on page 314).

The column information for the result set (SQLCINRD) may be totally contained in the block containing the OPNQRYRM or may overflow to subsequent blocks. If overflow occurs, a two-byte length field and the DDM code point of the SQLCINRD must follow the OBJDSS structure in each overflow block. The next byte of the SQLCINRD data follows the SQLCINRD code point.

The FD:OCA description of the answer set data may be totally contained in the block that ends the column information for the result set (SQLCINRD) or may overflow to subsequent blocks. If overflow occurs, a two-byte length field and the DDM code point (QRYDSC) of the FD:OCA description must follow the OBJDSS structure in each overflow block. The next byte of the FD:OCA description follows the QRYDSC code point.

BF4 The answer set data follows the FD:OCA description of the data.

For Limited Block Protocols, where none of the answer set columns is of a LOB data type, the answer set data may begin in the block that ends the FD:OCA description of the data, if space remains, filling up the block with as much answer set data as will fit into the block. It may be a partial row or multiple rows. If the application requester is capable of accepting extra blocks of answer set data, then the application server may chain additional blocks to the block that ends the FD:OCA description of the data. Alternatively, the block containing the description of the data can be sent without answer set data, deferring answer set data to the first CNTQRY command. An application server should not send extra blocks of answer set data if the extra blocks cannot either complete a row or contain at least one complete row.

For Limited Block Protocols, where some of the answer set columns are of a LOB data type, the answer set containing the last portion of the description of the data must be sent without any answer set data. Answer set data must be deferred to the first CNTQRY when an optional OUTOVR command data object from the application requester gives the desired format for the LOB data columns being returned in an application FETCH request.

For Fixed Row Protocols, the answer set containing the last portion of the description of the data must be sent without any answer set data. Answer set data must be deferred to the first CNTQRY command.

The first four bytes of the answer set data is a two-byte length field and the DDM code point (QRYDTA) for the answer set data. The entire answer set may be totally contained in the block in which the answer set begins or the answer set may overflow into subsequent blocks. If overflow occurs, a two-byte length field and the DDM code point (QRYDTA) for the answer set data must follow the OBJDSS structure in each overflow block. The next byte of answer set data follows the QRYDTA code point.

BF5 The two-byte length fields preceding the code points represent 4 bytes plus the length of the SQLCINRD data, the FD:OCA description, or the answer set data contained in the block in which the code point resides.

BF7 The four bytes containing the length and code point for QRYDSC, QRYDTA, or the SQLCARD, SQLDTARD, or SQLRSLRD of the summary component or the SQLCINRD for an EXCSQLSTT command (see query data transfer protocols rule QP2 in Section 7.19.2 on page 316) that invoked a stored procedure cannot be split across blocks.

BF8 No block may contain more than one QRYDSC code point nor more than one QRYDTA code point. A QRYDSC and a QRYDTA may both be contained in one OBJDSS or each may be in a separate OBJDSS.

BF9 An OPNQRYRM always begins a new block. When multiple blocks are sent as a response to an OPNQRY or a CNTQRY command, each standard block except the last block must be full. A short standard block means that it is the last standard block for the command. Only query blocks containing EXTDTAs (non-standard query blocks) may follow the short block. When multiple blocks are sent as a response to an EXCSQLSTT that invoked a stored procedure, each standard block must be full, except that the blocks of the summary component (see query data transfer protocols rule QP2 in Section 7.19.2 on page 316), a block preceding a block that contains an OPNQRYRM, or the last block may be short blocks.

BF10 An answer set row may span blocks.

BF12 When multiple blocks are sent in response to a single request (OPNQRY, CNTQRY, or EXCSQLSTT), the last DSS of one block must be chained to the first DSS of the next block.

BF13 If a standard block ends with a partial row of data that does not contain the end of that row, the partial row must completely fill the block.

BF14 The RDB Result Set Reply Message (RSLSETRM) of the summary component for an EXCSQLSTT command (see query data transfer protocols rule QP2 in Section 7.19.2 on page 316) that invoked a stored procedure may be totally contained in the first block of the result set or may overflow to subsequent blocks. If overflow occurs, a two-byte length field and the DDM code point of the RSLSETRM must follow the RPYDSS structure in each overflow block. The next byte of RSLSETRM data follows the RSLSETRM code point.

BF15 The SQLCARD of the summary component for an EXCSQLSTT command (see query data transfer protocols rule QP2 in Section 7.19.2 on page 316) that invoked a stored procedure may be totally contained in a single block of the result set or may overflow to a subsequent block. If overflow occurs, a two-byte length field and the DDM code point of the SQLCARD must follow the OBJDSS structure in the overflow block. The next byte of SQLCARD data follows the SQLCARD code point.

BF16 The SQLDTARD of the summary component for an EXCSQLSTT command (see query data transfer protocols rule QP2 in Section 7.19.2 on page 316) that invoked a stored procedure may be totally contained in a single block of the result set or may overflow to subsequent blocks. If overflow occurs, a two-byte length field and the DDM code point of the SQLDTARD must follow the OBJDSS structure in the overflow blocks. The next byte of SQLDTARD data follows the SQLDTARD code point.

BF17 The SQLRSLRD of the summary component for an EXCSQLSTT command (see query data transfer protocols rule QP2 in Section 7.19.2 on page 316) that invoked a stored procedure may be totally contained in a single block of the result set or may overflow to subsequent blocks. If overflow occurs, a two-byte length field and the DDM code point of the SQLRSLRD must follow the OBJDSS structure in the overflow blocks. The next byte of SQLRSLRD data follows the SQLRSLRD code point.

BF18 Answer set data consists of *base row data* and *externalized row data*.

Base row data for an answer set row includes each column in the answer set row, either as the column data itself or as an FD:OCA placeholder for the column data. If the base row contains an FD:OCA placeholder for a column, the column data flows in an associated EXTDTA object as externalized row data. A nullable column that is null has no associated EXTDTA. A column with a zero length has no associated EXTDTA.

All base row data flows in QRYDTA objects.

Externalized row data flows in EXTDTA objects, according to Block Format Rule BF19 (see Section 7.19.1.1 on page 310) and Chaining Rules CH3 and CH4 (see Section 7.19.1.3 on page 314).

An answer set row may consist of only base row data or may consist of base row data and externalized row data. A base row is *complete* when all the base row data for an answer set row has been sent. An answer set row consisting of only base row data is *complete* when the base row is complete; that is, when all columns in the answer set row have been sent to the application requester as base row data. An answer set row consisting of both base row data and externalized row data is *complete* when the base row is complete and all associated EXTDTAs for the row have been sent.

Only columns that are FD:OCA Generalized String may be externalized. All other data types must flow as base row data.

BF19 Blocking applies to all query reply messages and reply data objects except EXTDTA objects.

Each EXTDTA always begins a new non-standard query block. The query block consists of the complete EXTDTA object. No other query reply data or reply messages may be contained in the query block.

An EXTDTA is contained in a single DSS and a DSS containing an EXTDTA does not contain any other objects.

The complete query or result set response is composed of one or more blocks, each block consisting of one or more DSSs.

For a description of the possible block formats, including the description of answer set data and the reply messages for OPNQRY, CNTQRY, and EXCSQLSTT, see the DDM description of DDM terms LMTBLKPRC (Limited Block Protocols) and FIXROWPRC (Fixed Row Protocols).[43]

7.19.1.2 Block Size (BS Rules)

BS1 The application requester determines the block size for standard query blocks. The application server must send blocks equal to the application requester specified size. The exceptions are the last block sent in response to an OPNQRY or CNTQRY and the blocks of the summary component, a block preceding a block that contains an OPNQRYRM, or the last block sent in response to an EXCSQLSTT. These blocks may be short blocks (truncated). See rule BF9 in Section 7.19.1.1 on page 310, rule IR1 in Section 7.8 on page 294, and rules QT2, QT3, QT4, and QT5 in Section 7.19.3 on page 318.

43. Formerly known as SNGROWPRC in DDM Level 3 documentation.

Block size is an operand of OPNQRY, CNTQRY, and EXCSQLSTT. Block size may change on any or each CNTQRY request.

The block size does not apply to non-standard query blocks.

BS2 The minimum block size for standard query blocks is set at 512 bytes.

BS3 The maximum block size for standard query blocks is set at 32,767 bytes.

7.19.1.3 Chaining (CH Rules)

CH1 The DDM RPYDSS or OBJDSS chaining indicator must be used to chain multiple DSSs in the same block. The symbol RPYDSS_52 represents an RPYDSS with the chaining flag set (the DSSFMT is set to X'52'). The symbol OBJDSS_53 represents an OBJDSS with the chaining flag set (X'53') and OBJDSS_03 represents an OBJDSS without the chaining flag set. The only valid chaining configurations for a block are the following:

Note: Parentheses surround optional repeatable OBJDSS_53s.

- (RPYDSS_52) - (OBJDSS_53) - OBJDSS_03
- (RPYDSS_52) - (OBJDSS_53) - RPYDSS_52 - (OBJDSS_53) - OBJDSS_03
- (OBJDSS_53) - OBJDSS_03
- (OBJDSS_53) - RPYDSS_52 - (OBJDSS_53) - OBJDSS_03

The following examples illustrate the relationship between DDM objects and their requisite carriers. Table 7-1 is a maximal example that shows everything that can be specified, in order, in a response to an OPNQRY that consists of a single block.

Table 7-1 Maximal Example for OPNQRY

DDM OBJECT	DDM CARRIER
OPNQRYRM	RPYDSS
TYPDEFNAM	OBJDSS
TYPDEFOVR	OBJDSS
SQLCARD	OBJDSS
TYPDEFNAM	OBJDSS
TYPDEFOVR	OBJDSS
QRYDSC	OBJDSS
TYPDEFNAM	OBJDSS
TYPDEFOVR	OBJDSS
QRYDTA	OBJDSS
TYPDEFNAM	OBJDSS
TYPDEFOVR	OBJDSS
ENDQRYRM	RPYDSS
TYPDEFNAM	OBJDSS
TYPDEFOVR	OBJDSS
SQLCARD	OBJDSS

Table 7-2 on page 315 is a maximal example that shows everything that can be specified, in order, in the first block of the response to an EXCSQLSTT that returns one or more result sets.

Table 7-2 Maximal Example for the First Block of an EXCSQLSTT that Returns Result Sets

DDM OBJECT	DDM CARRIER
RDBUPDRM	RPYDSS
RSLSETRM	RPYDSS
TYPDEFNAM	OBJDSS
TYPDEFOVR	OBJDSS
SQLDTARD	OBJDSS
TYPDEFNAM	OBJDSS
TYPDEFOVR	OBJDSS
SQLRSLRD	OBJDSS

Each DDM object carried in an OBJDSS, except for EXTDTA objects, may be carried in its own OBJDSS or combined with other objects to reduce the number of OBJDSSs sent.

Each EXTDTA object always flows in its own OBJDSS. In Table 7-1 on page 314, if EXTDTAs are associated with the row or rows in the single query block, then each flows in its own OBJDSS following the QRYDTA object containing the last column of its containing row.

CH2 In all cases where more than one block of answer set data is returned in response to a single request, except for the last block, the last (or only) DDM DSS in a block is chained to the first (or only) DDM DSS in the next block.

CH3 The EXTDTA objects associated with a row cannot flow until the associated base row is complete (that is, all columns in the corresponding base row have been sent).

CH4 The EXTDTA objects associated with an answer set row flow in the same order that their corresponding FD:OCA placeholders appear in the base row.

All EXTDTA objects associated with a row must flow before other EXTDTAs from subsequent rows can flow.

If the AR can accept extra query blocks, all EXTDTA objects associated with the row or rows in a previous query block must flow before each subsequent extra query block can flow.

CH5 If a base data object contains FD:OCA placeholders, then the EXTDTA objects to be sent in the chain with the base data object must be chained in sequence after the base data object.

The only object that may be chained after an EXTDTA is either another EXTDTA object or a QRYDTA object for the same query.

If the command is a CNTQRY and the query is completed by the command, then the ENDQRYRM should not be put in the query block, even if there is room for it in the query block, if there are any EXTDTAs to be returned for the QRYDTA, either at the same time as the QRYDTA (if *rtnextall*) or with the next CNTQRY (if *rtnextrow*).

If the command is an EXCSQLSTT for a stored procedure call with query result sets, the EXTDTAs returned associated with the SQLDTARD containing the parameters must be the last objects in the reply chain. They follow the results set information objects and the query reply objects for the result sets. No extra query blocks may be returned with any query result sets in this case.

7.19.2 Query Data Transfer Protocols (QP Rules)

QP1 Fixed Row Protocol

In the non-error case, the response to OPNQRY consists of one or more blocks containing the OPNQRY reply message (OPNQRYRM) and the FD:OCA description of the data (QRYDSC). When multiple blocks are returned, the last DSS of block *n* is chained to the first DSS of block *n+1*. The chaining indicator is not set in the last block. No answer set data is included in the response.

The application requester must use CNTQRY to retrieve answer set data. Each use of CNTQRY retrieves exactly the number of rows of answer set data requested by the application, or the number of rows available in the answer set if more rows were requested than available during multi-row fetches.[44] The answer set is transmitted in one or more blocks; the number of blocks depends on the number of rows (multi-row fetches) returned, the size of the rows, and the number of EXTDTA objects to be sent. When multiple blocks are returned, the last DSS of block *n* is chained to the first DSS of block *n+1*. The chaining indicator is not set in the last block.

EXTDTA objects associated with retrieved rows are sent in accordance with Chaining Rules CH3 and CH4 in Section 7.19.1.3 on page 314.

For non-scrolling cursors, the query is complete when a CNTQRY results in a returned block containing an RPYDSS indicating end of query (ENDQRYRM) chained to an OBJDSS containing an SQLCARD data object. If answer set data is contained in the query block, then the query is complete when all EXTDTA objects associated with the base rows in the object have been returned. The EXTDTAs associated with the object must be received and processed before the subsequent reply message and objects in the query block.

For cursors that scroll, the query completes when the application closes the cursor. This results in a CLSQRY command flowing to the application server.

The DDM term FIXROWPRC more completely defines this protocol. See rules QT1, QT2, QT3, and QT4 in Section 7.19.3 on page 318.

QP2 Limited Block Protocol

In the non-error case, the response to OPNQRY consists of one or more blocks containing the OPNQRY reply message (OPNQRYRM) and the FD:OCA description of the data (QRYDSC). The block containing the end of the FD:OCA description may be completed, if room exists, with answer set data (adhering to Block Rules BF4, BF7, BF8, and BF9 in Section 7.19.1.1 on page 310). If the application requester is capable of accepting extra blocks of answer set data, then the application server may chain additional blocks to the block that ends the FD:OCA description of the data. When multiple blocks are returned, the last DSS of block *n* is chained to the first DSS of block *n+1*. The chaining indicator is not set in the last block.

In the non-error case, the response to an EXCSQLSTT that returns result sets consists of one or more blocks containing a summary component and one or more query result set components. The query result set components follow the summary component. The summary component consists of a a Result Set reply message (RSLSETRM), an

44. Multi-row fetch is not supported in DRDA Level 1.

SQLCARD or SQLDTARD, and an SQLRSLRD. Each result set component consists of the OPNQRY reply message (OPNQRYRM), the column name information for the result set (SQLCINRD), and the FD:OCA description of the data (QRYDSC). Within each result set component, the block containing the end of the FD:OCA description may be completed, if room exists, with answer set data (adhering to Block Rules BF4, BF7, BF8, and BF9 in Section 7.19.1.1 on page 310). Further, if the application server is capable of accepting extra blocks of answer set data, then the application server may chain additional blocks to the block that ends the FD:OCA description of the data taking into account Chaining Rule CH5. When multiple blocks are returned, the last DSS of block *n* is chained to the first DSS of block *n+1*. The chaining indicator is not set in the last block.

The application requester must use CNTQRY to retrieve more answer set data.

- If none of the answer set columns can flow as externalized FD:OCA data in EXTDTAs, each use of CNTQRY has a response consisting of as many blocks as necessary to contain the end of a row. The block containing the end of a row may be filled, if room exists, with additional answer set data. If the application requester is capable of accepting extra blocks of answer set data, then the application server may chain additional blocks to the block that ends the FD:OCA description of the data.

- If any of the answer set columns can flow as externalized FD:OCA data in EXTDTAs, the application requester specifies whether EXTDTA objects are to be sent a row at a time or whether all EXTDTA objects associated with returned query blocks are to be sent with the query blocks. Each use of CNTQRY has one of the following responses:

  — For the first CNTQRY, or a subsequent CNTQRY retrieving additional base row data, the application server returns as many query blocks as necessary to contain the end of a base row. The block containing the end of a row may be filled, if room exists, with additional base row data. If the application requested that all EXTDTAs be returned with the base data, then the EXTDTA objects for all complete base rows in the command response are returned as non-standard query blocks following that query block. The response is complete. The next CNTQRY command retrieves additional base row data along with any associated EXTDTAs.

    If EXTDTA objects are to be returned a row at a time, no EXTDTAs are returned with the base data. The response is complete. The next CNTQRY command retrieves the EXTDTAs for the first base row for which there are associated EXTDTAs. The application requester does not send a CNTQRY to retrieve EXTDTAs if a base row has only null placeholders or placeholders with zero lengths. After all base rows previously sent have been completed with any associated externalized data, the next CNTQRY command retrieves additional base row data.

  — For a subsequent CNTQRY retrieving externalized row data associated with a complete base row previously sent, the application server returns query blocks containing the EXTDTAs corresponding to the FD:OCA placeholders in the base row. This rule applies only if EXTDTAs objects are to be returned a row at a time.

When multiple blocks are returned, the last DSS of block *n* is chained to the first DSS of block *n+1*. The chaining indicator is not set in the last block.

The query or result set is complete when a CNTQRY, OPNQRY, or an EXCSQLSTT results in a returned block containing an RPYDSS indicating end of query (ENDQRYRM) chained to an OBJDSS containing an SQLCARD data object. The RPYDSS may or may not be chained from an OBJDSS containing the last row of answer set data. If answer set data contained in the query block has any associated EXTDTAs that are to be returned with a subsequent CNTQRY, then Chaining Rule CH5 applies to the RPYDSS.

The DDM term LMTBLKPRC more completely defines this protocol. See rules QT1, QT2, QT3, and QT4 in Section 7.19.3.

An attempt to UPDATE WHERE CURRENT OF CURSOR or DELETE WHERE CURRENT OF CURSOR on a cursor that is fetching rows using the limited block protocol results in an SQLSTATE of 42828.

QP3 The OPNQRY reply message (OPNQRYRM) indicates whether the application server is using Fixed Row Protocols or Limited Block Protocols for the query or result set.

7.19.3 Query Data or Result Set Transfer (QT Rules)

QT1 The application server terminates an open query or result set when it receives and processes a CLSQRY command or when it detects other conditions that implicitly closed the cursor. Any time an implicit close occurs, one of the following reply messages must be sent:

- ENDQRYRM: normal end of answer set data
- ABNUOWRM: RDB initiated Rollback

An OBJDSS containing an SQLCARD data object follows each of these messages.

A terminated query is the same as a query that has not yet been opened.

QT2 Each query terminating reply message (RPYDSS) must be chained to, and can only be chained to, an OBJDSS carrying an SQLCARD data object. The SQLCARD may contain additional information describing the reason for query termination.

For example, the reply message ABNUOWRM may be chained to an SQLCARD data object that carries the name of a resource involved in a deadlock that generated a relational database rollback operation.

QT3 The OBJDSS carrying the SQLCARD data object returned with a query terminating reply message must be chained from the terminating reply message RPYDSS, must be contained in the same response block as the RPYDSS, and must be the last item in the response block.

QT4 The RPYDSS representing the query terminating reply message must be the first item in the response block in the following cases:

- When the query data transfer protocol is Fixed Row with a single row fetch.
- When the query data transfer protocol is Limited Block and the reply message is ABNUOWRM—RDB initiated Rollback.

  See Section 7.8 on page 294 for a description.

In all other cases, the query terminating reply message RPYDSS may, if room exists in the response block, be chained from the OBJDSS containing the last row of answer set data, taking into account Chaining Rule CH5 if it applies.

QT5 When the query terminating reply message RPYDSS containing an ENDQRYRM and the associated SQLCARD data object OBJDSS are placed into a separate block following the above rules, then the block can be sent only in response to a subsequent CNTQRY command.

See rule IR1 in Section 7.8 on page 294 for the handling of the DDM reply message ABNUOWRM.

7.19.4 Additional Query and Result Set Termination Rules

The following section provides additional rules for terminating queries and result sets within DRDA flows. The objective of these rules is to avoid the CLSQRY request/response message exchange between application requester and application server when possible and to keep cursor states consistent between application requester and application server. The rules are in figures that show a set of conditions and actions to be taken for the conditions. Each row of the figure represents a condition or an action. Each column of the figure represents a set of conditions and the actions that apply to the set of conditions. Thus, each column represents a specific case. Each case is described in narrative form following the figure that contains the case. For readability, the conditions and actions are separated and each column has a unique identifier.

For example, in Table 7-3 on page 320, column H has the conditions that the cursor is open, a CNTQRY command has been received, the SQL FETCH returned an end-of-data SQLCA, the query is without the HOLD option, and room exists in the current block for the ENDQRYRM RPYDSS chained to an SQLCARD OBJDSS. The actions are to perform an SQL CLOSE cursor, mark the cursor as not open, create and place the ENDQRYRM/SQLCARD in the block, and send the block to the application requester.

7.19.4.1 Rules for OPNQRY, CNTQRY, CLSQRY, and EXCSQLSTT

Table 7-3 AS Rules for OPNQRY, CNTQRY, CLSQRY

	Cases																
Conditions	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**
CURSOR STATE:																	
NOT OPEN	A			D										N		P	
OPEN		B			E	F	G	H	I	J	K	L	M		O		
DRDA COMMAND:																	
CNTQRY	A	B					G	H	I	J	K	L					
CLSQRY				D	E	F							M				
OPNQRY														N	O	P	
EXCSQLSTT																	Q
OPEN CURSOR FAILED																P	
SQL FETCH RETURNED SQLCA																	
AND DATA ROW OR MULTI-ROWS							G										
SQL FETCH RETURNED SQLCA																	
WITHOUT A DATA ROW																	
QUERY TERMINATING																	
OTHER								H	I								
ROLLBACK										J	K						
BLOCK IS EMPTY											K						
NON-QUERY TERMINATING												L					
ROOM IN BLOCK FOR																	
RPYDSS / SQLCARD																	
YES								H									
NO									I								
RPYDSS / SQLCARD STACKED		B			E	F											
ENDQRYRM						F											
OTHER					E												

	Cases																
Actions	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**
RETURN SQLCARD:																	
RC='00000'						F											
FROM RDB								H					M			P	
CURSOR STATE:																	
NOT OPEN	A	B		D	E	F		H			K		M			P	
OPEN							G		I	J		L		N	O		Q
ISSUE SQL CLOSE CURSOR								H	I				M				
PLACE RPYDSS / SQLCARD IN BLOCK								H									
STACK RPYDSS / SQLCARD									I								
SEND ROLLBACK RPYDSS / CA											K						
STACK ROLLBACK RPYDSS / CA										J							
SEND STACKED RESPONSE		B			E												
PURGE STACK		B			E	F											
SEND BLOCK						F		H	I	J		L	M				
PROCESS CNTQRY							G										
SEND QRYNOPRM RPYDSS	A			D													
PROCESS OPNQRY														N			
PROCESS EXCSQLSTT																	Q
SEND QRYPOPRM RPYDSS															O		
SEND OPNQFLRM RPYDSS / CA																P	

Cases for Rules

CASE A The application server cursor state indicates that the cursor is not open. The request from the application requester is a CNTQRY. The application server returns a QRYNOPRM reply message indicating the cursor is not open. The cursor state remains not open.

CASE B The application server cursor state indicates that the cursor is open. The request from the application requester is a CNTQRY. The application server determines that an RPYDSS and SQLCARD are stacked pending receipt of an application requester request. There are two stacks to consider. This first stack would be the reply message ENDQRYRM. See rules QT1, QT2, QT3, and QT4 in Section 7.19.3 on page 318. The other stack applies to all operations. The stacked reply message would be ABNUOWRM. See rules QT1, QT2, QT3, and QT4 in Section 7.19.3 on page 318, and rule IR1. The latter stack is always the first to be processed (for all application requester requests, regardless of whether the request is for a query operation or otherwise). ENDQRYRM applies only to non-scrolling cursors. The cursor positioning for cursors that scroll can be moved to any row in the answer set on the next FETCH. Therefore, the cursor should never be pre-closed by the application server.

When an ABNUOWRM / SQLCARD is stacked, this response is sent for the received request. The stack is purged. All cursor states are set to not open, and all cursor stacks are purged (these cursor actions could have taken place when the ABNUOWRM was created and placed upon the stack).

When an ENDQRYRM/SQLCARD is stacked on a cursor, and when the next request is a CNTQRY, this response is sent for the received request. The stack is then purged. The cursor state is set to not open.

CASE D The application server cursor state indicates that the cursor is not open. The request from the application requester is a CLSQRY. No pending responses are stacked. The application server returns a QRYNOPRM reply message indicating the cursor is not open. The cursor state remains not open.

CASE E The application server cursor state indicates that the cursor is open. The request from the application requester is CLSQRY. The application server determines that an ABNUOWRM and SQLCARD are stacked, pending receipt of an application requester request. The stacked response is sent for the received request. The stack is purged, and the cursor state is set to not open.

CASE F The application server cursor state indicates that the cursor is open. The request from the application requester is CLSQRY. The application server determines that an ENDQRYRM and SQLCARD are stacked, pending receipt of the next application requester request for this cursor. This is a normal condition. The application wants to close the cursor before reaching end of data.

In this situation, the application server has already reached end of data. Rather than sending the stacked ENDQRYRM, the application server sends an SQLCARD with an SQLSTATE of 00000. The cursor stack is purged, and the cursor state is set to not open.

CASE G The application server cursor state indicates that the cursor is open. The request from the application requester is a CNTQRY. The process CNTQRY action is taken, which is tailored to the query data transfer protocol in effect for the cursor. See rules QP1 and QP2 in Section 7.19.2 on page 316.

The SQL FETCH condition is a result of the PROCESS CNTQRY action. The CNTQRY process also includes the process for spanning answer set rows across multiple blocks and sending EXTDTA objects in non-standard query blocks. This is not shown in the diagram.

CASE H The application server cursor state indicates that the cursor is open. The request from the application requester is a CNTQRY. The process CNTQRY action resulted in a relational database query terminating response to a FETCH request. In DRDA, all query terminating responses to a FETCH for queries without the HOLD option, are mapped to ENDQRYRM except a rollback, which is mapped to ABNUOWRM.

In CASE H, we have an ENDQRYRM condition with enough room in the current block for the ENDQRYRM reply message chained to an SQLCARD. The SQLCARD will contain the SQLSTATE from the relational database. The RPYDSS/SQLCARD is appended to any data already in the block. The OBJDSS in the block, if one exists, is chained to the RPYDSS, which is chained to the SQLCARD/OBJDSS. The application server closes the cursor by requesting the relational database to close the cursor. The application server then sets the cursor state to not open and sends the block.

CASE H also applies to the ABNUOWRM query terminating condition for fixed row protocols.

This case does not apply if the command returns any base data for which there are associated EXTDTAs. See CASE I instead.

CASE I This is the same as CASE H except there is no room in the current block for the RPYDSS/SQLCARD. The block is sent to the application requester without the RPYDSS/SQLCARD (a short block). The RPYDSS/SQLCARD is stacked on the cursor waiting for the next request. The cursor state remains open.

This case also applies if the command returns any base data for which there are associated EXTDTAs.

CASE J This is the same as CASE I except the query terminating condition is a rollback (ABNUOWRM). Because of rule QT4 (see Section 7.19.3 on page 318), the ABNUOWRM must be the first DSS in the block. Therefore, if the block contains any data, the ABNUOWRM/SQLCARD must be stacked waiting for the next request from the application requester. The current block, with the accumulated answer set data, is sent, and the cursor state remains open.

CASE K This is the same as CASE J when the current block is empty. Therefore, the application server does not stack the ABNUOWRM/SQLCARD but instead sends this response to the application requester. All cursors are set to the not open state.

CASE L The application server cursor state indicates that the cursor is open. The request from the application requester is a CNTQRY. The process CNTQRY action resulted in a FETCH request that returned an SQLCA without a data row, and the application server has determined that this is not a query terminating condition. This means the relational database can accept a subsequent FETCH. DRDA does not define these conditions. The SQL semantic for FETCH as communicated by SQLSTATEs determines these conditions, if they exist or if they will ever exist.

The DRDA-defined action for these conditions is for the application server to interrupt the process of filling the block. That is, the application server returns a short block to the application and waits for the next request. If any completed base rows in the short block included FD:OCA placeholders for externalized data, the associated EXTDTAs are returned according to rules QP1 and QP2 in Section 7.19.2 on page 316. The application may decide to issue another FETCH, resulting in CNTQRY, close the cursor, resulting in CLSQRY, or rollback or terminate. The application requester is not dependent upon nor sensitive to these conditions. Therefore, the application server returns the SQLCA that the relational database has provided as QRYDTA and with a null data row.

CASE M The application server cursor state indicates that the cursor is open. The request from the application requester is CLSQRY. The application server requests the relational database to close the cursor. The application server sets the cursor state to not open and returns an SQLCARD to the application requester. The SQLCARD is derived from the SQLCA that the relational database has returned for the SQL close cursor operation.

CASE N The application server cursor state indicates that the cursor is not open. The request from the application requester is OPNQRY. The application server performs the OPNQRY process, which is not described. The cursor state is set to open.

CASE O The application server cursor state indicates that the cursor is open. The request from the application requester is OPNQRY. The cursor state remains open. The application server returns the QRYPOPRM reply message indicating the cursor is already open.

CASE P The application server cursor state indicates that the cursor is not open. The request from the application requester is OPNQRY. The OPEN CURSOR fails. The application server returns the reply message OPNQFLRM chained to the SQLCARD. The cursor state remains not open.

CASE Q The request from the application requester is an EXCSQLSTT that invokes a stored procedure that returns one or more result sets. The application server executes the stored procedure, which is not described. The cursor state for each result set is set to open.

7.19.4.2 Rules for FETCH

Table 7-4 AR Rules for FETCH

	Cases																			
Conditions	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**
CURSOR STATE:																				
NOT OPEN	A																			
CLOSE ONLY		B																		
OPEN			C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	
POSITION IN BLOCK:																				
SQLCA / ROW				D	E									N	O	P				
SQLCA / NULL ROW OR ROW / SQLSTATE>02999						F	G	H									Q	R	S	
RPYDSS									I	J	K									
END OF BLOCK												L	M							
CHAINING:																				
CHAINED TO DSS:																				
THIS BLOCK			C			F			I					N			Q			
NEXT BLOCK				D			G			J		L			O			R		
NOT CHAINED					E			H			K		M			P			S	
MULTI_ROW FETCH:																				
YES														N	O	P	Q	R	S	
NO			C	D	E	F	G	H												
DOESN’T MATTER:	A	B							I	J	K	L	M							

	Cases																			
Actions	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**
RETURN SQLCA:																				
EOQ - SQLSTATE=’02000’		B																		
NOT OPEN	A																			
FROM AS QRYDTA			C	D	E	F	G	H												
FROM AS SQLCARD									I											
BUILT BY AR														N	O	P	Q	R	S	
CURSOR STATE:																				
NOT OPEN	A																			
CLOSE ONLY		B							I											
OPEN			C	D	E	F	G	H				L	M	N	O	P	Q	R	S	
RETURN ROW OR MULTI-ROWS			C	D	E									N	O	P				
ISSUE CNTQRY - RCV ALL BLOCKS													M							
GET NEXT RECEIVED BLOCK												L								
PROTOCOL ERROR										J	K									

Cases for Rules

CASE A The application requester cursor state indicates that the cursor is not open. The application requester, therefore, returns a cursor not open SQLCA to the application and leaves the cursor state as not open.

CASE B The application requester cursor state indicates that the only valid operation against the cursor is to close it. This is because the application server has processed to end of data and has returned the ENDQRYRM and SQLCARD. The application has probably issued another FETCH after having received the end of data SQLCA. The application requester, therefore, returns another end of data SQLCA (SQLSTATE 02000) and leaves the cursor state as close only.

CASE C The application requester cursor state indicates that the cursor is open, not performing a multi-row fetch, and the application requester is positioned in the block to an answer set row with the associated SQLCA (may be null). The OBJDSS is chained to another DSS within the current block, but chaining is not relevant. The application requester, therefore, returns the row and associated SQLCA to the application and positions itself to the next data item in the block. The cursor state remains OPEN.

The RETURN ROW OR MULTI-ROWS action includes the process of a row spanned across blocks. If the row is a base row having FD:OCA placeholders, the RETURN ROW OR MULTI-ROWS action includes the process of retrieving the externalized data associated with each FD:OCA placeholder from an EXTDTA object. If the EXTDTA objects for the query are being returned a row at a time, the application requested must issue a CNTQRY command to receive the associated EXTDTA objects. Retrieving the externalized data for the base row does not change the application requester's position in the query block being processed. These processes are not shown in the diagram.

CASE D This is the same as CASE C; chaining is not relevant.

CASE E This is the same as CASE C; chaining is not relevant.

CASE F The application requester cursor state indicates that the cursor is open and that the application requester is positioned in the block to a null answer set row and a non-null SQLCA, or a non-null answer set row and the SQLSTATE is greater than 02999. The application requester therefore returns the SQLCA to the application and positions itself to the next data item in the block. The cursor state remains open; chaining is not relevant.

CASE G This is the same as CASE F; chaining is not relevant.

CASE H This is the same as CASE F; chaining is not relevant.

CASE I The application requester cursor state indicates that the cursor is open, and that the application requester is positioned in the block to an RPYDSS, which is chained to an SQLCARD in the current block. The application requester, therefore, returns an SQLCA to the application, which is derived from the SQLCARD. The cursor state is set to the close only state, meaning that the only valid operation against the cursor is to close the cursor.

Note: If the RPYDSS is an ABNUOWRM, all cursor states are placed in the NOT OPEN state. The rollback has reset all cursors. All buffers associated with the cursors are reset to an empty state.

CASE J The application requester cursor state indicates that the cursor is open and that the application requester is positioned in the block to an RPYDSS, which is chained to an SQLCARD in the next block. This is a protocol error.

CASE K The application requester cursor state indicates that the cursor is open and that the application requester is positioned in the block to an RPYDSS, which is not chained to an SQLCARD. This is a protocol error.

CASE L The application requester cursor state indicates that the cursor is open and that the application requester is positioned at the end of the block. The last OBJDSS in the block is chained to a DSS in the next block. The application requester receives the next block, leaves the cursor state at OPEN, and positions itself at the beginning of the new block. Then the application requester re-evaluates conditions based on the data found in the next block.

CASE M The application requester cursor state indicates that the cursor is open and that the application requester has positioned itself at the end of the block. The last OBJDSS in the block is not chained. Therefore, the application requester must issue CNTQRY and then receive the next block. Then the application requester re-evaluates conditions based on the data found in the next block.

CASE N The application requester cursor state indicates the cursor is open, performing a multi-row fetch,[45] and the application requester is positioned to an answer set row with the associated SQLCA (which may be null).

The answer set row is the *n*th row, where *n* is less than or equal to the number of rows requested in the FETCH statement but not less than the number of contiguous rows beginning at the first row requested up to the first terminating condition. The application requester returns to the application a block of rows and a statement-level SQLCA constructed from information in all of the SQLCAs associated with the returned rows. The process of building the statement-level SQLCA and returning associated rows is terminated by one of the following conditions:

- All rows from the application server are processed.

  In this case, the application requester positions itself at the end of the block.

- A row (possibly null) with an associated SQLSTATE greater than 02999 is encountered. Processing of this row is deferred to the next FETCH.

  In this case, the application requester positions itself at the error row.

- An RPYDSS is encountered. Processing of the RPYDSS is deferred to the next FETCH.

  In this case, the application requester positions itself at the RPYDSS.

Chaining is not relevant. The cursor state remains open.

The RETURN ROW OR MULTI-ROWS action includes the process of a row spanned across blocks. If the row is a base row having FD:OCA placeholders, the RETURN ROW OR MULTI-ROWS action includes the process of retrieving the externalized data associated with each FD:OCA placeholder from an EXTDTA

45. Multi-row fetch is not supported in DRDA Level 1.

object. If the EXTDTA objects for the query are being returned a row at a time, the application requested may need to issue a CNTQRY command to retrieve the associated EXTDTA objects. Retrieving the externalized data for the base row does not change the application requester's position in the query block being processing. These processes are not shown in the diagram.

CASE O This is the same as CASE N; chaining is not relevant.

CASE P This is the same as CASE N; chaining is not relevant.

CASE Q The application requester cursor state indicates that the cursor is open, performing a multi-row fetch, and that the application requester is positioned in the block to a null answer set row and a non-null SQLCA or a non-null answer set row and the SQLSTATE is greater than 02999. The application requester returns to the application a statement-level SQLCA reflecting the error condition for the single row. The row, if present, is not returned to the application. The application requester positions itself to the next data item in the block. The cursor state remains open; chaining is not relevant.

CASE R This is the same as CASE Q; chaining is not relevant.

CASE S This is the same as CASE Q; chaining is not relevant.

7.19.4.3 Rules for CLOSE

Table 7-5 AR Rules for CLOSE

	Cases															
Conditions	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**
CURSOR STATE:																
NOT OPEN	A															
CLOSE ONLY		B														
OPEN			C	D	E	F	G									
RPYDSS IN RCVD BLOCKS:																
YES			C	D	E											
NO						F	G									
RPYDSS CHAINED TO SQLCARD OBJDSS IN CURRENT BLOCK:																
YES			C													
NO				D												
RPYDSS CHAINED TO UNRECEIVED BLOCK					E											
OBJDSS CHAINED TO UNRECEIVED BLOCK							G									

	Cases															
Actions	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**
RETURN SQLCA:																
RC=’00000’		B	C													
NOT OPEN	A															
FROM AS						F										
CURSOR STATE:																
NOT OPEN	A	B	C			F										
ISSUE CLSQRY						F										
PROTOCOL ERROR				D	E		G									

Cases for Rules

CASE A The application requester cursor state indicates the cursor is not open. The application requester, therefore, returns an SQLCA with a not open SQLSTATE of 24501. The cursor state remains not open.

CASE B The application requester cursor state indicates that the only valid operation against the cursor is to close it. This is because the application server has processed to end of data and has returned the ENDQRYRM and SQLCARD. The application requester, therefore, returns an SQLCA with an SQLSTATE of 00000 to the application.

CASE C The application requester cursor state indicates that the cursor is open. A query terminating RPYDSS is in the current block and is chained to an SQLCARD in the same block. The application requester returns an SQLCA to the application with an SQLSTATE of 00000 and sets the cursor state to not open.

CASE D The application requester cursor state indicates that the cursor is open. An RPYDSS is in the current block but does not indicate chaining. This is a protocol error.

CASE E The application requester cursor state indicates that the cursor is open. An RPYDSS is in the current block but is chained to a DSS in the next block. This is a protocol error.

CASE F The application requester cursor state indicates that the cursor is open. The current block does not contain a query-terminating RPYDSS. This is a normal situation. The application wants to close the cursor prior to viewing all the answer set data, and the application server has not reached end of data. The application requester, therefore, must issue a CLSQRY command to the application server and wait for the reply. When the application requester receives the reply, it places the cursor in the not open state and returns the SQLCA, which is derived from the SQLCARD returned to the application by CLSQRY.

CASE G The application requester cursor state indicates that the cursor is open. The current block does not contain an RPYDSS, but the last OBJDSS in the block is chained to the next block. This is a protocol error, a violation of rule QP1 or QP2 (see Section 7.19.2 on page 316).

In Fixed Row Protocols, the last FETCH operation would have received all blocks the application server sent and would have processed all the data in the blocks.

In Limited Block Protocols, the last operation, either a FETCH or OPEN, would have received all the blocks the application server sent. The last block received may have contained data the next operation would process, but the block would not have been chained.

Chapter 8

SQLSTATE Usage

This chapter identifies the SQLSTATEs that DRDA specifically references. See *ISO/IEC 9075: 1992, Database Language SQL* for definitions of SQLSTATEs referenced in DRDA as well as other SQLSTATEs appropriate for error conditions not defined or not within the scope of DRDA.

This chapter also identifies the SQLSTATEs that an application program receives following the receipt of a DDM reply message at an application requester in response to a DRDA remote request that the application requester made on behalf of the application program.

This chapter also provides a general description for each SQLSTATE that any other chapter of this volume references.

8.1 DRDA Reply Messages and SQLSTATE Mappings

Table 8-1 lists the valid DDM reply messages and *svrcod*s for DRDA. The table is also a mapping between the reply messages and SQLSTATEs. If an application requester receives a valid reply message with a valid *svrcod*, the application requester must return the SQLSTATE listed in the table. If an application requester receives a reply message that is not valid in DRDA or a valid reply message with an *svrcod* that is not valid in DRDA, the application requester returns the SQLSTATE 58018.

Reply messages CMDVLTRM, CMMRQSRM, and RDBUPDRM are not supported in DRDA Level 1.

Table 8-1 DRDA Reply Messages (RMs) and Corresponding SQLSTATEs

REPLY MESSAGE	SVRCOD	SQLSTATE
ABNUOWRM	8	SQLSTATE in SQLCARD
ACCRDBRM	0	00000
ACCRDBRM	4	01539
AGNPRMRM	16,32,64	58009
BGNBNDRM	8	SQLSTATE in SQLCARD
CMDATHRM	8	58008 or 58009
CMDCHKRM	0	Not returned
CMDCHKRM	8	58008 or 58009
CMDCHKRM	16,32,64,128	58009
CMDNSPRM	8	58014
CMDVLTRM	8	58008
CMMRQSRM	8	Not returned or 2D528 or 2D529
DSCINVRM	8	58008 or 58009
DTAMCHRM	8	58008 or 58009

REPLY MESSAGE	**SVRCOD**	**SQLSTATE**
ENDQRYRM	4,8	SQLSTATE in SQLCARD
ENDUOWRM	4	SQLSTATE in SQLCARD
MGRDEPRM	8	58009
MGRLVLRM	8	58010
OBJNSPRM	8	58015
OPNQFLRM	8	SQLSTATE in SQLCARD
OPNQRYRM	0	SQLSTATE in SQLCARD
PKGBNARM	8	58012
PKGBPARM	8	58011
PRCCNVRM	8	58008 or 58009
PRCCNVRM	16,128	58009
PRMNSPRM	8	58016
QRYNOPRM	4	24501
QRYNOPRM	8	58008 or 58009
QRYPOPRM	8	58008 or 58009
RDBACCRM	8	58008 or 58009
RDBATHRM	8	08004
RDBNACRM	8	58008 or 58009
RDBAFLRM	8	SQLSTATE in SQLCARD
RDBNFNRM	8	08004
RDBUPDRM	0	Not returned
RSCLMTRM	8,16	57012
RSCLMTRM	32,64,128	57013
RSLSETRM	0	SQLSTATE in SQLCARD
SECCHKRM	0	Not returned
SECCHKRM	8,16	42505
SQLERRRM	8	SQLSTATE in SQLCARD
SYNTAXRM	8	58008 or 58009
TRGNSPRM	8	58008 or 58009
VALNSPRM	8	58017

8.2 SQLSTATEs that DRDA References

01515 This SQLSTATE reports a warning on a FETCH or SELECT into a host variable list or structure that occurred because the host variable was not large enough to hold the retrieved value. The FETCH or SELECT does not return the data for the indicated SELECT item, the indicator variable is set to −2 to indicate the return of a NULL value, and processing continues.

01519 This SQLSTATE reports an arithmetic exception warning that occurred during the processing of an SQL arithmetic function or arithmetic expression that was in the SELECT list of an SQL select statement, in the search condition of a SELECT or UPDATE or DELETE statement, or in the SET clause of an UPDATE statement. For each expression in error, the indicator variable is set to −2 to indicate the return of a NULL value. The associated data variable remains unchanged, and processing continues.

01520 A string value cannot be assigned to a host variable because the value is not compatible with the host variable.

This SQLSTATE reports a translation warning (no representation of the character in the application requester CCSID) that may occur when translating a string value the application server returned to the application requester. The string value cannot be assigned to a host variable that has an indicator variable within a SELECT statement of an application program. The value is incompatible with the host variable due to a mismatch in data representation. The FETCH or SELECT does not return the data for the indicated SELECT item, the indicator variable is set to −2 to indicate the return of a NULL value, and processing continues.

01539 This SQLSTATE reports a *character set restriction* exception warning. The connection is established, but only the single byte character set (SBCS) is supported. Any attempted usage of the restricted CCSIDs results in an error.

01587 This SQLSTATE reports a pending response or a mixed outcome from at least one participant during the two-phase process.

01615 Bind option ignored.

The bind operation continues. The first ignored option is reported in SQLERRMC.

02000 This SQLSTATE reports a *No Data* exception warning due to an SQL operation on an empty table, zero rows identified in an SQL UPDATE or SQL DELETE statement, or the cursor in an SQL FETCH statement was after the last row of the result table.

08001 The Application Requester is unable to establish the connection.

This SQLSTATE reports the failure of an attempt to make a DRDA connection. If the associated SQLCODE is −30082, the failure was related to DRDA security protocols. In that case, reason code 2 in the related message tokens specifies the detailed cause of the failure.

08004 Server not found or server authorization failure

This SQLSTATE reports that a user attempted to access a relational database that cannot be found or that a user is not authorized to access the relational database.

0A501 This SQLSTATE reports a failure to establish a connection to an application server. This SQLSTATE should be used if the security mechanism specified by the application server is not supported by the application requester.

22001 This SQLSTATE reports an error on a FETCH or SELECT into a host variable list or structure that occurred because the host variable was not large enough to hold the retrieved value. The FETCH or SELECT statement is not executed. No data is returned.

22003, 22012, 22502, or 22504
This SQLSTATE reports an arithmetic exception error that occurred during the processing of an SQL arithmetic function or arithmetic expression that was in the SELECT list of an SQL select statement, in the search condition of a SELECT or UPDATE or DELETE statement, or in the SET clause of an UPDATE statement. The statement cannot be executed. In the case of an INSERT or UPDATE statement, no data is updated or deleted.

22021 The value of a string host variable cannot be used as specified or a string value cannot be assigned to a host variable because the value is not compatible with the host variable.

This SQLSTATE reports a conversion error (no representation for a character in the application server CCSID) that may occur when converting an application input string variable to the application server's representation. The value of the string host variable is incompatible with its use due to a mismatch in data representation. The value cannot be used as specified.

This SQLSTATE reports a conversion error (no representation of the character in the application requester CCSID) that may occur when converting a string value the application server returned to the application requester. The string value cannot be assigned to a host variable that does not have an indicator variable within a SELECT statement of an application program. The value is incompatible with the host variable due to a mismatch in data representation. The FETCH or SELECT statement is not executed. No data is returned. If the statement was a FETCH, then the cursor remains open.

24501 Execution failed due to an invalid cursor state. The identified cursor is not open.

25000 Operation invalid for application execution environment.

This SQLSTATE reports the attempt to use SQL update operations to change data within a relational database in a read-only application execution environment.

2D521 SQL COMMIT or ROLLBACK statements are invalid in the current environment.

This SQLSTATE reports the attempt to execute an SQL commit or rollback process in an environment that does not allow SQL COMMIT or ROLLBACK statements.

2D528 Operation invalid for application execution environment.

This SQLSTATE reports the attempt to use EXCSQLIMM or EXCSQLSTT to execute a COMMIT in a dynamic COMMIT restricted environment.

2D529 Operation invalid for application execution environment.

This SQLSTATE reports the attempt to use EXCSQLIMM or EXCSQLSTT to execute a ROLLBACK in a dynamic ROLLBACK restricted environment.

40504 Unit of Work Rolled Back.

This SQLSTATE reports that the unit of work rolled back due to a system error. This SQLSTATE is not used during commit processing.

42505 This SQLSTATE reports a failure to authenticate the end user during connection processing to an application server.

42828 This SQLSTATE reports an attempt to DELETE WHERE CURRENT OF CURSOR or UPDATE WHERE CURRENT OF CURSOR on a cursor that is fetching rows using a blocking protocol.

42932 Program preparation assumptions are incorrect.

This SQLSTATE reports that the program preparation assumptions in effect for a BNDSQLSTT command are incorrect.

51021 Application must execute rollback.

SQL statements cannot be executed until the application process executes a rollback operation.

56084 An unsupported SQLTYPE was encountered in a select-list or input-list.

This SQLSTATE reports that an SQL statement cannot be processed because of an unsupported SQLTYPE. This error can occur when a sender detects an SQLTYPE that cannot be sent to the receiver because the receiver is at an SQLAM level lower than the minimum level at which the SQLTYPE is supported. The sender rejects the statement with this SQLSTATE and the data is not sent. This error can also occur when a receiver at a given SQLAM level detects an SQLTYPE that is supported at that SQLAM level, but for which it does not provide support and for which there is no compatible mapping according to Data Conversion rules (DC3 to DC5). The receiver rejects this statement.

56095 Invalid bind option.

This SQLSTATE reports that one or more bind options were not valid at the server. The bind operation terminates. The first bind option in error is reported in SQLERRMC.

56096 Conflicting bind options.

The bind operation terminates. The bind options in conflict are reported in SQLERRMC.

57012 Execution failed due to unavailable resources that will not affect the successful execution of subsequent commands or SQL statements.

This SQLSTATE reports insufficient target resources that are non-relational database resources.

57013 Execution failed due to unavailable resources that will affect the successful execution of subsequent commands or SQL statements.

This SQLSTATE reports insufficient target resources that are non-relational database resources.

57014 This SQLSTATE reports the successful interrupt of a DRDA request.

57017 This SQLSTATE reports a lack of support for data conversion. Execution failed because the CCSIDs required for data conversion are unsupported.

58008 Execution failed due to a distribution protocol error that will not affect the successful execution of subsequent commands or SQL statements.

This SQLSTATE reports a DRDA protocol error that causes termination of processing for a specific DRDA command or SQL statement.

Each of these errors is a programming error.

58009 Execution failed due to a distribution protocol error that caused deallocation of the conversation.

This SQLSTATE reports a DRDA protocol error that causes termination of processing for a specific command or SQL statement. When an application requester returns this SQLSTATE, the application requester must also deallocate the conversation on which the application server reported the protocol error.

Each of these errors is a programming error.

58010 Execution failed due to a distribution protocol error that will affect the successful execution of subsequent commands or SQL statements.

This SQLSTATE reports a DRDA protocol error that causes termination of processing for a specific command or SQL statement and for any subsequent DRDA commands and SQL statements that the application program issued.

A manager level not supported error may not be a programming error.

58011 Command invalid while bind process in progress.

This SQLSTATE reports an attempt to execute a specific DRDA DDM command that is not valid while a Bind process is in progress. BNDSQLSTT, ENDBND, RDBCMM, and RDBRLLBCK are the only legal commands while a Bind process is in progress.

58012 Bind process with specified package name and consistency token not active.

This SQLSTATE reports an attempt to execute a BNDSQLSTT or ENDBND for a bind process that was not active.

58014 Command not supported error.

This SQLSTATE reports that the target does not support a particular command. The error causes termination of processing of the command, but does not affect the processing of subsequent DRDA commands and SQL statements that the application program issued.

58015 Object not supported error.

This SQLSTATE reports that the target does not support a particular object. The error causes termination of processing of the command, but does not affect the processing of subsequent DRDA commands and SQL statements that the application program issued.

58016 Parameter not supported error.

This SQLSTATE reports that the target does not support a particular parameter. The error causes termination of processing of the command, but does not affect the processing of subsequent DRDA commands and SQL statements that the application program issued.

58017 Value not supported for parameter.

This SQLSTATE reports that the target does not support a particular parameter value. The error causes termination of processing of the command, but does not affect the processing of subsequent DRDA commands and SQL statements that the application program issued.

58018 Reply message with not supported error.

This SQLSTATE reports the receipt of an RM with an RM code point that DRDA does not recognize or with an *svrcod* value that DRDA does not recognize. The error does not affect the processing of subsequent DRDA commands and SQL statements that the application program issued.

The cause of this error may be a mismatch in source and target manager levels or may be a programming error.

58028 Unit of Work Rolled Back.

This SQLSTATE reports that the unit of work rolled back when it was requested to commit. The rollback occurred as a result of a resource not capable of committing. This SQLSTATE does not guarantee that all resources rolled back.

Open Group Technical Standard

Part 2:

Environmental Support

The Open Group

Chapter 9

Environmental Support

Part 1, Database Access Protocol discusses the core of the architecture that makes it what it is, a distributed relational database architecture. But this alone does not describe all that is needed to provide a robust distributed relational database environment. This section describes the characteristics of various components in a distributed environment that are necessary to provide a robust environment that supports access to distributed relational databases. These components are:

- Communications
- Security
- Accounting
- Transaction Processing
- Problem Determination

Part 3, Network Protocols discusses these components when implemented for specific network protocols.

9.1 DDM Communications Model and Network Protocol Support

The key component of the DDM communications model is the DDM communications manager. The DDM communications manager provides the following functions:

- Interfaces with local network facilities to receive and send DDM requests, replies, and data
- Routes received DDM requests and replies to the appropriate agent
- Accepts requests, replies, and data from an agent and packages them into the proper data stream format for transmission
- Detects normal and abnormal termination of network connections and responds in an appropriate fashion

For further detail, refer to the DDM term CMNMGR in the DDM Reference.

The purpose of the DDM communications model is to provide a conceptual framework for viewing DRDA communications. DRDA, however, does not require that the communications components of DRDA implementing products replicate the structure of the DDM communications model. DRDA does require that the communications components of DRDA implementing products implement DRDA request and response protocols.

DRDA does not require any particular network protocol, such as LU 6.2, TCP/IP, NetBIOS, for flowing the DRDA protocol. DRDA does specify the network protocol must provide certain characteristics that are required to provide robust support for a distributed relational database environment. These characteristics are:

- Timely communication outage notification
- Guaranteed in order and complete delivery of network messages
- Propagation of information that allows both sides of the connection to identify the partner

The communication protocol might also provide additional functionality that could be used to support the environment. Examples of this additional functionality are:

- Propagation of security and accounting information
- Propagation of synchronization point processing information

9.2 Accounting

DRDA requires the ability to acquire information useful for accounting. This information is categorized as who, what, when, and where information. The who information is the end-user name and it is provided through the network protocols or through DRDA mechanisms as defined in identification and authentication processing. The what information is provided in some of the network protocols or can be found in the DRDA-defined correlation token that is passed on ACCRDB. The when information is provided by locally available clocks. The where information is provided by mechanisms that extract the unique network identifier for the participants in the network connection.

9.3 Transaction Processing

Transaction processing in DRDA is the process to commit or rollback a unit of work across one or multiple application servers involved in the unit of work. DRDA works in cooperation with the network protocols and synchronization point managers to provide this support. If a network protocol does not support the two-phase commit process, then application servers that are connected on those protocols have operational restrictions as defined by DRDA (see Section 4.4.12.2 on page 120).

Chapter 10

Security

DRDA requires the ability to identify and authenticate the end user associated with the DRDA requests. Some network protocols such as LU 6.2, provide the ability to pass the information necessary to identify and authenticate the end user. See Part 3, Network Protocols for a description of this capability for the specific network protocols.

Not all network protocols provide this capability. For environments where this is the case, DRDA defines DDM flows for passing security information (see Section 4.4.2 on page 61). DRDA provides the ability for the application requester and application server to negotiate the security mechanisms to use to provide the identification and authentication support. These mechanisms are described in this chapter.

10.1 DCE Security Mechanisms with GSS-API

DRDA provides support for utilizing The Open Group's DCE security mechanisms. This section briefly describes the flows that perform identification and authentication through GSS-API with DCE security. The description of GSS-API uses the Generic Security Services Application Programming Interface (GSS-API). An implementation may choose another interface as long as it is compatible with GSS-API.

Figure 10-1 provides a greatly simplified overview of the flows involved with calling GSS-API to utilize DCE security mechanisms. The actual DCE processing to perform the identification and authentication processing is described in the DCE documentation listed in **Referenced Documents** on page xxiv. Following the figure is a description of the flows.

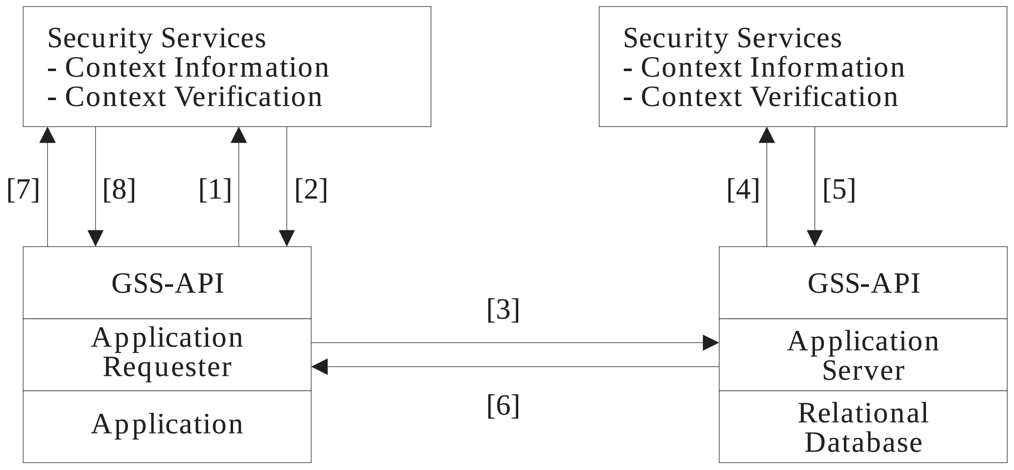


Figure 10-1 Using GSS-API to Call DCE-Based Security Flows in DRDA

1 The application makes a request that requires access to the application server. Acting on behalf of the end user of the application, the application requester calls the security services (*gss_init_sec_context*( )) in order to obtain security context information for accessing the application server. In this example, the application requester requests mutual authentication by setting the **gss_c_mutual_flag** to true on the *gss_init_sec_context*( ) call.

2 The security services return to the application requester, a major_status code of GSS_S_CONTINUE_NEEDED and security context information to be passed to the application server. The major_status code value indicates the security services processing is not complete and the application requester will receive security context information from the application server which will need to be passed to the security services to continue processing.

3 The application requester passes the security context information to the application server using a SECCHK command and a SECTKN object.

4 The application server calls the security services (*gss_accept_security_context*( )) to process the security context information.

5 The security services return to the application server, a major_status code of GSS_S_COMPLETE and security context information to be returned to the application requester. The major_status code value indicates the security services processing is complete and authentication of the application requester is successful.

6 The application server passes the security context information to the application requester using a SECCHKRM and a SECTKN object.

7 The application requester calls the security services (*gss_init_security_context*( )) to process the security context information.

8 The security services returns a major_status code value of GSS_S_COMPLETE indicating the security services processing is complete and authentication of the application server is successful.

10.2 Userid-Related Security Mechanisms

DRDA provides the following userid-related security mechanisms:

- Userid and password
- Userid, password, and new password
- Userid-only
- Userid and password substitute
- Userid and encrypted password

The following sections provide overviews of these mechanisms.

10.2.1 Userid and Password Security Mechanism

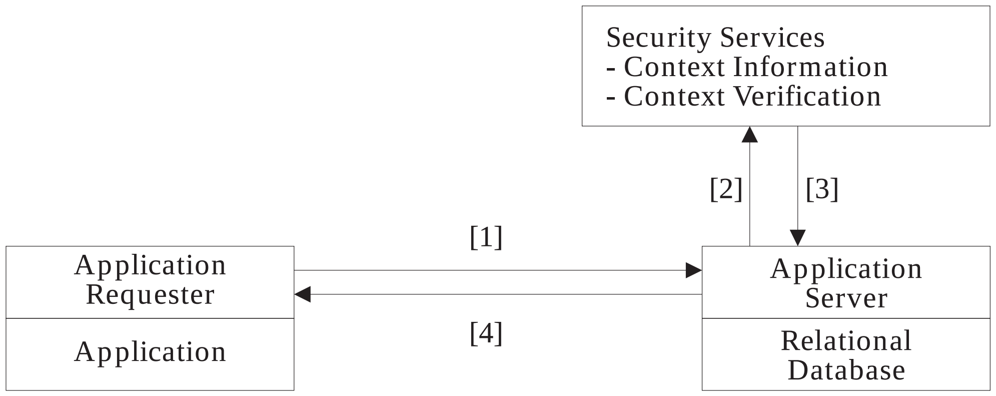


Figure 10-2 Userid and Password Authentication Flows

The following description of the flows does not define the interface between the application server and the security services. It is assumed that local services are available at the application server to accept the userid and password and authenticate the userid based on this information.

1 The application makes a request that requires access to the application server. The application requester acquires a password for the end user that is associated with the application. The process to acquire the password is platform-specific. The application requester passes the userid and password to the application server in the *usrid* and *password* parameters on SECCHK.

2 The application server calls the security services to process the userid and password.

3 The security services returns an indication the end user is authenticated.

4 The application server returns a SECCHKRM to the application requester indicating the authentication is successful.

10.2.2 Userid, Password, and New Password Security Mechanism

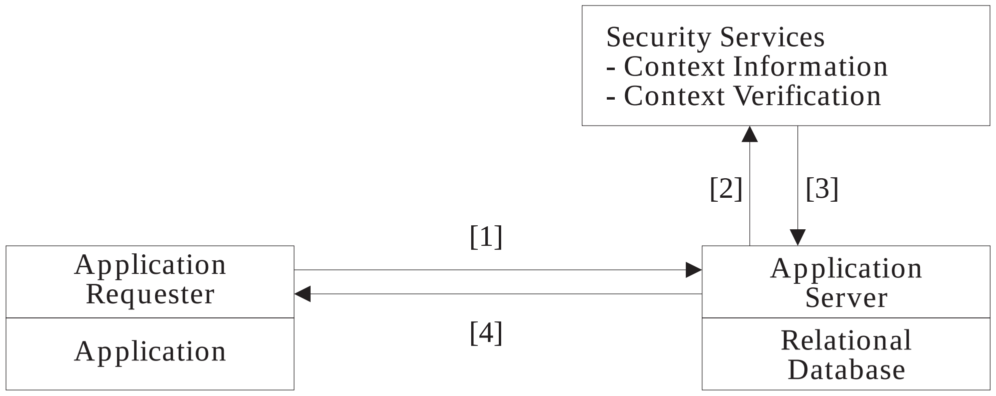


Figure 10-3 Userid, Password, and New Password Authentication Flows

The following description of the flows does not define the interface between the application server and the security services. It is assumed that local services are available at the application server to accept the userid, password, and new password and to authenticate the userid and the changing of the password based on this information.

1 The application makes a request that requires access to the application server. The application requester acquires a password and new password for the end user that is associated with the application. The process to acquire the passwords is platform-specific. The application requester passes the userid, password, and new password to the application server in the *usrid*, *password*, and *newpassword* parameters of SECCHK.

2 The application server calls the security services to process the userid and passwords.

3 The security services returns an indication that the end user is authenticated and that the password has been replaced.

4 The application server returns a SECCHKRM to the application requester indicating that the authentication and the changing of the password is successful.

10.2.3 Userid-Only Security Mechanism

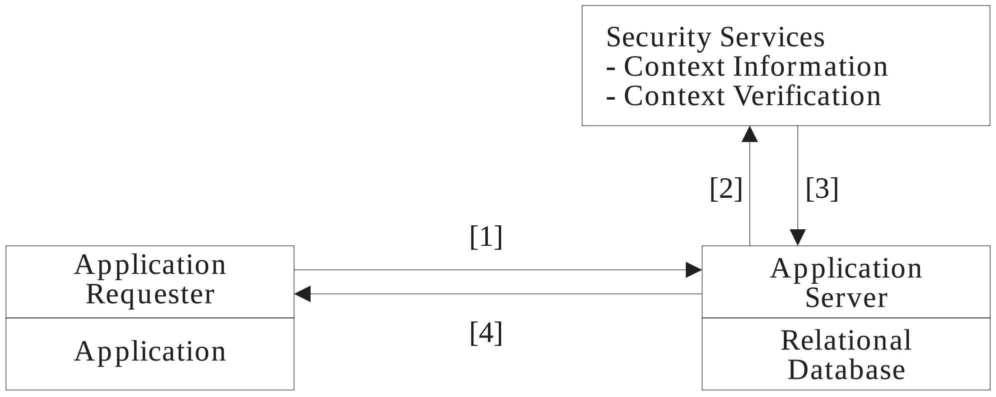


Figure 10-4 Userid and Password Authentication Flows

The following description of the flows does not define the interface between the application server and the security services. It is assumed that local services are available at the application server to accept the userid and password and authenticate the userid based on this information.

1 The application makes a request that requires access to the application server. A password is not required between the application requester and application server, so the application requester passes the userid to the application server in the *usrid* parameter on SECCHK.

2 The application server calls the security services to process the userid.

3 The security services return an indication the end user is authenticated.

4 The application server returns a SECCHKRM to the application requester indicating the authentication is successful.

10.2.4 Userid and Password Substitute Security Mechanism

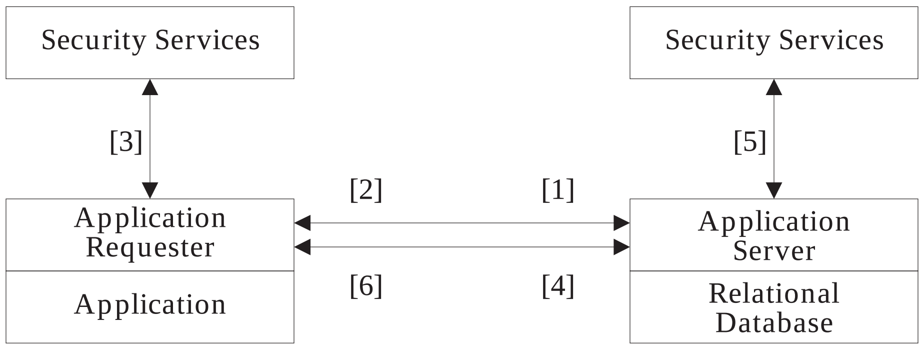


Figure 10-5 Userid and Password Substitute Authentication Flows

The following description of the flows does not define the interface between the application server and the security services. It is assumed that local services are available at the application requester and at the application server to perform the required functions described below:

1 The application makes a request that requires access to the application server. The application requester sends an ACCSEC command with SECMEC value of USRSBSPWD and SECTKN containing eight bytes of random data (application requester's) seed.

2 The application server saves the client's seed and replies with ACCSECRD containing the server's seed in SECTKN which also consists of eight bytes of random data.

3 The application requester acquires a password for the end user that is associated with the application. The process to acquire the password is platform-specific. The application requester then creates a password substitute using the two seeds, the clear text password, and the Data Encryption Standard (DES) algorithm, following the procedure described in the PWDSBS term in the DDM Reference.

4 The application requester passes the userid and the password substitute to the application server in the USRID and PASSWORD parameters on SECCHK.

5 The application server creates a password substitute of its own from its knowledge of the seeds and the password to be validated. It compares the values it computes to that which it received from the application requester. If they match, the user is validated.

6 The application server returns a SECCHKRM to the application indicating success or failure based on the outcome of the authentication process.

10.2.5 Userid and Encrypted Password Security Mechanism

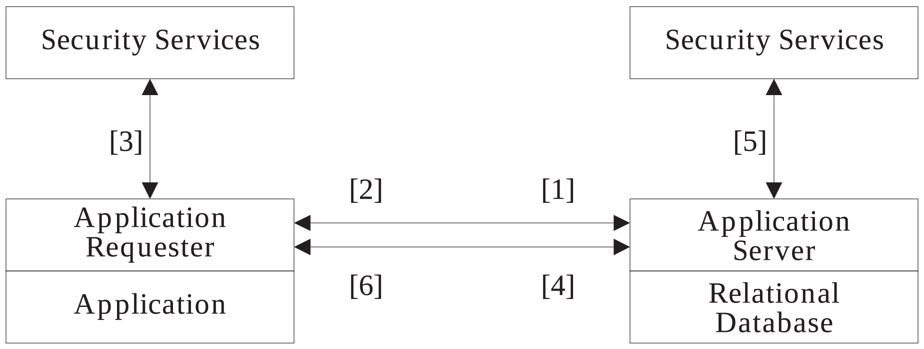


Figure 10-6 Userid and Encrypted Password Authentication Flows

The following description of the flows does not define the interface between the application server and the security services. It is assumed that local services are available at the application requester and at the application server to perform the required functions described below.

1 The application makes a request that requires access to the application server. The application requester sends an ACCSEC command with SECMEC value of USRENCPWD and SECTKN containing the application requester's connection key, which is generated using the standard Diffie-Hellman key distribution algorithm to generate a shared private key. See the PWDENC term in the DDM Reference.

2 The application server saves the client's key and replies with ACCSECRD containing the server's connection key in SECTKN which also is generated using the Diffie-Hellman algorithm.

3 The application requester acquires a password for the end user that is associated with the application. The process to acquire the password is platform-specific. The application requester then encrypts the password using the DES password, userid, and generated Diffie-Hellman shared private key described in the DDM Reference.

4 The application requester passes the userid and the encrypted password to the application server in the USRID and PASSWORD parameters on SECCHK.

5 The application server decrypts the encrypted password using the DES password, userid, and generated Diffie-Hellman shared private key. It then asks the local security subsystem to validate the userid/password combination.

6 The application server returns a SECCHKRM to the application indicating success or failure based on the outcome of the authentication process.

Chapter 11

Problem Determination

The DRDA environment involves remote access to relational database management systems. Because the access is remote, enhancements to the local problem determination process were needed. These enhancements use network management tools and techniques. DRDA-provided enhancements are messages to focal points and a standard display for the correlation token value. In DRDA, the correlator between focal point messages and locally generated diagnostic information is the ACCRDB *crrtkn* parameter value.

In a remote unit of work environment, an application accesses only one database management system at a time, so the requester can easily track the failing component when things go wrong. The existing tools, which work for local applications, should be adequate for debugging most of the problems. The DRDA tools and techniques discussed in this chapter enhance the process for problem determination.

In distributed unit of work environment, an application accesses multiple database management systems at the same time. An SQL statement can only operate on one database management system at a time, and the application uses SQL connection management to indicate which database management system is currently active. As in DRDA Level 1, a requester can determine how to proceed when errors occur, so the current tools should be adequate.

For background information on this topic, see the references listed in **Referenced Documents** on page xxiv.

11.1 Network Management Tools and Techniques

In addition to local tools, the DRDA environment should use the following tools or techniques for problem determination and isolation. The use of these tools are recommended; however, use of them is not mandatory.

11.1.1 Standard Focal Point Messages

A standard focal point message (that is, SNA alert) provides a generic format for reporting problem-related information. This structure is flexible enough to report errors from all different operating environments and is able to communicate with the network management program in the environment where the error occurred.

11.1.2 Focal Point

A focal point is a consistent destination for all problem-related information. To operate in a DRDA environment, a focal point can be beneficial because it provides a single point to view problems. This point provides support personnel with all the information to solve a problem or decide on the proper steps to get more information. A logical focal point would be a network management program like Netview or Netview/PC. The focal point would need the ability to talk with all other network management programs participating in the distributed environment.

11.1.3 Correlation

Since a single problem might be related to work at multiple sites, a correlator value is needed to tie the problem together as a single related problem. DRDA defines a correlation value for this.

The correlation value needs to be unique to avoid value collisions with other non-related units of work. DRDA takes advantage of the inherent uniqueness of a network address and adds a time stamp value to this string to provide uniqueness within that address.

The generic format of the correlation value exchanged when an application requester is accessing an RDB is as follows:

```
Generic CRRTKN format:
    x.yz  where x is 1 to 8 bytes (variable), character
                y is 1 to 8 bytes (variable), character
                z is 6 bytes (fixed), binary
          with a period (".") to delimit x from y, the total byte
          count is a variable between 9 and 23.
```

The x.y positions represent the network address and the z position is used to create uniqueness, of which a clock value might be used. In some cases, a unit of work identifier might fall into this format, and is therefore a valid correlation value.

The specific values of each field are dependent on where the work started which might include a non-DRDA environment. See Section 12.8.1 on page 414 and Section 13.6.1 on page 433 for the specifics when the values are generated at an application requester in a particular network environment.

It is also possible that a DRDA component will inherit a correlation value from some other source. If that value conforms to the format defined by DRDA, then it is used as the correlation value. Otherwise, the DRDA component must create a correlation value and provide a means to map to the inherited value.

11.2 DRDA Required Problem Determination and Isolation Enhancements

This section describes the DRDA requirements regarding correlation displays and collecting diagnostic information.

11.2.1 Correlation Displays

Because the correlation value is used to correlate information across multiple sites, it is important that a standard display of the correlation value is defined. The following are the rules for displaying a correlation value:

1. The generic display of the correlation value is as follows:

```
Displaying a CRRTKN value:
    x.y.z  where x is 1 to 8 bytes (variable), character
                 y is 1 to 8 bytes (variable), character
                 z is 12 bytes (fixed), character
           with periods (".") to delimit between x, y, and z
           total byte count is variable between 16 and 30.

SNA example:    NET.LU.123456789ABC
TCP/IP example: 09155467.9704.01234567689AB
```

11.2.2 DRDA Diagnostic Information Collection and Correlation

There is the need to:

- Collect supporting data for an error condition
- Correlate between focal point messages and supporting data

11.2.2.1 Data Collection

When an error condition occurs at an application requester or application server, data should be gathered at that location. The data collection process should use the current tools available for the local environment. An application requester and application server should collect diagnostic information when it receives a reply message (RM) or generates an RM listed in Table 11-1 on page 356. The application requester should gather diagnostic information when the network connection is unexpectedly dropped such as an LU 6.2 DEALLOCATE with a type of ABEND in an SNA environment.

11.2.2.2 Correlation Between Focal Point Messages and Supporting Data

Correlation between focal point messages and supporting data at each location, as well as cross location, is done through correlation tokens. In DRDA, the correlation token is the ACCRDB *crrtkn* parameter value. The *crrtkn* value can be inherited at the application requester from the operating environment. If the inherited value matches the format of the DRDA-defined correlation token, then it is sent at ACCRDB in the *crrtkn* parameter. If the application requester does not inherit a correlation value, or the value does not match the format of a DRDA-defined correlation token, then the application requester must generate a DRDA-defined correlation token. The correlation value is required in focal point messages and supporting diagnostic information.

11.3 Generic Focal Point Messages and Message Models

This section discusses focal point messages in support of the environments in which DRDA might be installed. There are several architectures that support focal point messages. Two of these architectures are SNA Management Services Generic Alerts and Simple Network Management Protocol (SNMP). Although these architectures are pervasive in the network environment they were developed for (Alerts for SNA, SNMP for TCP/IP), they are not restricted to those network environments. For example, SNA alerts might be used in a TCP/IP network, hence alert models defined in DRDA are usable in multiple network environments.

The following message models assume the use of SNA alerts in the environment.

11.3.1 When to Generate Alerts

It is recommended that alerts be generated when the following conditions exist. Some of these conditions have alert models defined for them. See Section 11.3.3 on page 355 for an example of condition to alert model mappings.

- DRDA alerts must be generated whenever something happens that changes the availability of database management system resources, or threatens to.
- Alerts must be generated for serious errors where intervention by an operator (rather than a correction by a user) is required to correct the situation.
- Programming and protocols errors should be alerted.
- Alerts generated when supporting data about an error condition is collected. The alert will point to this data.
- Security subversion attempts such as the identified reuse of the security context information received in a SECTKN object.

11.3.2 Alerts and Alert Structure

The following sections describe the required alerts for conditions encountered at the application server and application requester. The alerts for DRDA use the Generic Alert Architecture as a model for the alert structure. The following figures define the subvectors, subfields, and code points required.

The references listed in **Referenced Documents** on page xxiv should be used to gain a more complete understanding of the architecture of generic alerts.

11.3.2.1 Alert Implementation Basics

The *SNA Management Services: Alert Implementation Guide* (SC31-6809, IBM) is a good starting point for understanding the architecture of generic alerts. By categorizing the subvectors and subfields using who, what, where, when, and why, an architect or implementer can be sure to cover the needed information. Alerts should be recorded in a place available for support or operations personnel to see and take action on. Figure 11-1 on page 355 categorizes the subvectors, subfields, and code points used for DRDA.

WHO		
Subvector	**Subfield**	**Description**
X'10'		Product Set Identifier
X'11'		Product Identifier
	X'08'	Product Number
	X'04'	Version, Release, Modification
	X'06'	Product Common Name

WHAT		
Subvector	**Subfield**	**Description**
X'92'		Generic Alert Data

WHERE		
Subvector	**Subfield**	**Description**
X'05'		Hierarchy / Resource List
	X'10'	Hierarchy Name List
	X'11'	Associated Resource List

WHEN		
Subvector	**Subfield**	**Description**
X'01'		Date / Time
	X'10'	Local Date / Time

WHY		
Subvector	**Subfield**	**Description**
X'93'		Probable Causes
X'96'		Failure Causes
	X'01'	Failure Causes
	X'81'	Recommended Actions
	X'85'	Detailed Data
X'48'		Supporting Data Correlation
	X'85'	Detailed Data (Supporting data ptr)

Figure 11-1 Summary of Required Subvectors and Subfields

11.3.3 Error Condition to Alert Model Mapping

The following sections define the specific error condition to alert model mapping. The tables do not define all possible error conditions. When an error condition requires an alert, but an alert model is not defined, an appropriate alert should be generated from the alert models defined here.

11.3.3.1 Specific Alert to DDM Reply Message Mapping

Table 11-1 on page 356 defines the alert constructs to DDM reply messages (RM). The numbers following the RM in column one are the severity codes (*svrcods*) of the RMs. The column labeled *where* is the location in which the alert is to be generated. For each DDM RM created at the application server, the specified alert must be generated at the application server. For each DDM RM received at the application requester, the specified alert is generated at the application requester.

See the DDM Reference for a list of DDM reply messages and their accompanying severity codes.

Table 11-1 Alerts Required for DDM Reply Messages

DDM RM	Where	Alert Model	Additional Information
AGNPRMRM	AR/AS	AGNPRM (see Table 11-3 on page 358)	See alert model and the DDM Reference for information on DDM reply message AGNPRMRM.
CMDCHKRM 8,16,32,64,128	AR/AS	CMDCHK (see Table 11-6 on page 364)	See alert model and the DDM Reference for information on DDM reply message CMDCHKRM.
CMDVLTRM 8	AR/AS	CMDVLT (see Table 11-7 on page 365)	See alert model and the DDM Reference for information on DDM reply message CMDVLTRM.
DSCINVRM 8	AR/AS	DSCERR (see Table 11-8 on page 366)	See alert model and the DDM Reference for information on DDM reply message DSCINVRM.
DTAMCHRM 8	AR/AS	DSCERR (see Table 11-8 on page 366)	See alert model and the DDM Reference for information on DDM reply message DTAMCHRM.
PRCCNVRM 8,16,128	AR/AS	PRCCNV (see Table 11-10 on page 368)	See alert model and the DDM Reference for information on DDM reply message PRCCNVRM.
QRYNOPRM 8	AR/AS	QRYERR (see Table 11-11 on page 369)	See alert model and the DDM Reference for information on DDM reply message QRYNOPRM.
QRYPOPRM 8	AR/AS	QRYERR (see Table 11-11 on page 369)	See alert model and the DDM Reference for information on DDM reply message QRYPOPRM.
RDBNACRM 8	AR/AS	RDBERR (see Table 11-12 on page 370)	See alert model and the DDM Reference for information on DDM reply message RDBNACRM.
RDBACCRM 8	AR/AS	RDBERR (see Table 11-12 on page 370)	See alert model and the DDM Reference for information on DDM reply message RDBACCRM.
RSCLMTRM 16,32,64,128	AS	RSCLMT (see Table 11-13 on page 371)	See alert model and the DDM Reference for information on DDM reply message RSCLMTRM.
RSCLMTRM 8	AS	RSCLMT (see Table 11-13 on page 371)	Alert Type in subvector X'92' defined as X'12' for unknown. See alert model and the DDM Reference for information on DDM reply message RSCLMTRM.

DDM RM	Where	Alert Model	Additional Information
SECCHKRM 16	AR/AS	SECVIOL (see Table 11-14 on page 373)	See alert model and the DDM Reference for information on DDM reply message SECCHKRM.
SYNTAXRM 8	AR/AS	SYNTAX (see Table 11-15 on page 374)	See alert model and the DDM Reference for information on DDM reply message SYNTAXRM.

11.3.3.2 *Additional Alerts at the Application Requester*

Any blocking or chaining violations on data received at the application requester from the application server should be alerted. Any Data Stream Structure (DSS) errors on data received at the application requester from the application server should be alerted. A DEALLOCATE with a type ABEND (abnormal end) without an accompanied RM, should be alerted. Resource limits reached at the application requester should also be alerted. Table 11-2 defines the alert models to be used for these conditions.

Table 11-2 Additional Alerts Required at Application Requester

Condition	Alert Model	Additional Information
Resource Limits Reached	RSCLMT (see Table 11-13 on page 371)	Alert Type in subvector X‘92’ defined as X‘12’ for unknown. See alert model and the DDM Reference for information on DDM reply message RSCLMTRM.
Blocking Protocol Error	BLKERR (see Table 11-4 on page 362)	See blocking rules, Section 7.19.1.1 on page 310.
Chaining Violation	CHNVIO (see Table 11-5 on page 363)	See chaining rules, Section 7.19.1.3 on page 314.
DEALLOCATE type ABEND received from the application server without an accompanying DDM reply message from the application server	GENERR (see Table 11-9 on page 367)	See *SNA Transaction Programmer’s Reference Manual for LU Type 6.2* (GC30-3084, IBM) for more information on DEALLOCATE ABEND.
DSS error: Error in the Data Stream Structure received from the application server	SYNTAX (see Table 11-15 on page 374)	See the DDM Reference for more information on Data Stream Structures.

11.3.3.3 *DRDA-Defined Alert Models*

The next series of tables are models of alert categories DRDA uses. Table 11-1 on page 356 and Table 11-2 refer to these tables. The tables map alertable conditions to the model, and indicate further enhancements to the model, if necessary. Following Table 11-3 on page 358 is a description of the subvectors, subfields, and code points. Because the majority of the subvectors, subfields, and code points are common, the subsequent tables reference Table 11-3 on page 358 and add additional information if needed.

Alert Model AGNPRM

This alert model is for permanent agent error conditions.

Table 11-3 Alert Model AGNPRM

Alert ID Number		X‘2E0AA333’
Alert Type	X‘01’	Permanent
Alert Description	X‘2102’	Distributed Process Failed
Probable Causes	X‘1050’	Agent Program
User Causes	(none)	
Install Causes	(none)	
Failure Causes	X‘1050’ X‘F0A3’	Agent Error Failure Occurred On (sf85)
Actions	X‘32D1’	Report The Following Logical Unit Of Work Identifier (sf85)(sf85)(sf85)
	X‘00B0’	Perform Problem Determination Procedure For (sf85)
	X‘00E1’	Perform (sf83) Problem Determination Procedures
	X‘0500’	Run Appropriate Trace
	X‘2203’	Review Supporting Data At Alert Sender
	X‘30E1’	Contact Service Representative For (sf83)
	X‘32D0’	Report The Following (sf85)(sf85)(sf85)
	X‘32A0’	Report The Following (sf85)
	...	...
Additional SVs	X‘10’ SV	Product Set Identifier
	X‘11’ SV	Product Identifier
	X‘08’ SF	Product Number
	X‘04’ SF	Version, Release, Modification
	X‘06’ SF	Product Common Name
	X‘05’ SV	Hierarchy / Resource List
	X‘10’ SF	Hierarchy Name List
	X‘11’ SF	Associated Resource List
	X‘01’ SV	Date / Time
	X‘10’ SF	Local Date / Time
	X‘48’ SV	Supporting Data Correlation
	X‘85’ SF	Detailed Data (Supporting data ptr)
	X‘47’ SV	MSU Correlation
	X‘20’ SF	CRRTKN (LUWID or UOWID format)
	...	...

The following descriptions of the fields in the above alert are only a summary. For a more complete description, see the **Referenced Documents** on page xxiv.

Alert ID Number

The Alert Identification Number is a field in subvector X‘92’.

Alert Type

The Alert Type is a field in subvector X‘92’. X‘01’ in this model defines a permanent loss of availability of the resource.

Alert Description

The Alert Description is a field in subvector X‘92’. It is a code point to define what has failed.

Probable Causes

The Probable Causes subvector is X‘93’. This subvector isolates the problem to a particular component or process.

User Causes and Install Causes

DRDA does not require the User Causes and Install Causes subvectors.

Failure Causes

The Failure Causes subfield is X‘01’. This subfield is used with the Failure Causes subvector X‘96’. This subvector and subfield relate the occurrences that might have happened to the process or component listed in the Probable Causes subvector. The subfield X‘85’ in the Failure Causes code point X‘F0A3’ should contain the following data beginning at byte 7. The subfield X‘85’ uses the data ID code point X‘0087’ for relational database. The detailed data field contains the actual RDB_NAME of the target relational database.

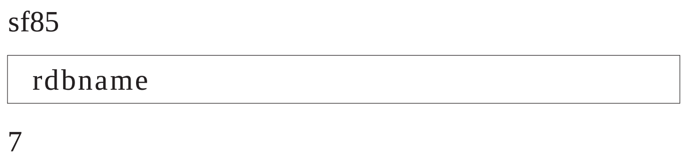


Figure 11-2 Subfield X‘85’ for Failure Causes Code Point X‘F0A3’

Actions

The Recommended Actions subfield is X‘81’. This subfield is used in conjunction with subvector X‘96’. This subfield defines the recommended actions for this error condition. A list of code points define the recommended actions. The implementing products should choose the code points that best fit their environment. Those code points should be listed in the order of priorities with the most important coming first followed by the next most important.

Action code point X‘32D0’ should be used to report symptom string information if it is available.

DRDA requires code point X‘32D1’. This code point displays the SNA LUWID or DDM UOWID. For DRDA Level 1 connections, the LUWID value is used for correlating the supporting data. The format for the LUWID or UOWID should follow the format defined for the long form of the display as defined in Section 12.8.1.2 on page 414. The three subfield X‘85’s in this code point should be sent in the following order with the following data. The data ID code point X‘0000’ should be used for all three subfield X‘85’s. In Figure 11-3 the first subfield X‘85’ contains the NETID.LUNAME or IPADDR.PORT portion of the LUWID or UOWID followed by a period. The second subfield X‘85’ contains the instance number, followed by a period and sequence number. The third subfield X‘85’ is blank and needs to be coded as a blank.

sf85

netid.luname.

7

sf85

|instance.sequence

7

sf85

7

Figure 11-3 Subfield X‘85’s for Actions Code Point X‘32D1’

DRDA Level 2 requires code point X‘32A0’ if the *crrtkn* is available. This code point displays the *crrtkn* value, which is the format of an unprotected LUWID, and is used for correlating supporting data. The subfield X‘85’ contains the correlation value with the data ID code point of X‘0101’ for correlation ID. Figure 11-4 displays the subfield X‘85’ that is associated with this code point.

sf85

netid.luname.instance

7

Figure 11-4 Subfield X‘85’ for Actions Code Point X‘32A0’

Additional SV

The following subvectors and subfields are additional subvectors and subfields required in the alert. They provide miscellaneous information to enhance the alert.

- Subvectors X‘10’ and X‘11’

  These subvectors and the accompanying subfields define the resource that is in error. There might be two subvector X‘10’s in an alert. The first one is for the alert sender. The second one is for the resource that is experiencing the problem. If the resource experiencing the problem is the same as the resource sending the alert, then only one subvector X‘10’ is present.

- Subvector X‘05’

  This subvector and accompanying subfields are used to provide a map of the unit of work. There should be two resource names defined in subfield X‘10’. These resource names are:

1. *AR* to represent the application requester. The code point for resource type identifier should be X'42' for requester.
2. *AS* to represent application server. The code point for resource type identifier should be X'43' for server.

Following subfield X'10' is subfield X'11'. This subfield is a list of associated resources for the application requester and application server. These resources should be defined by preceding the actual resource with an application requester or application server. For example, the relational database related to an application server would be defined as *AS rdbname*, the user ID related to the application requester on VM would be *AR vmid*.

Do not use subvector X'04' (SNA Address List) in the alerts. The use of subvector X'05' (Hierarchy Resource List) in conjunction with subfield X'11' (Associated Resource List) allows the display of the logical components for the failing unit of work.

- Subvector X'01'

  This subvector and its accompanying subfield provide a date and time for the alert. The optional extension of time field should be used for two bytes, which allows a 1/65535 fraction of a second.

- Subvector X'48'

  This subvector and its accompanying subfield is used as a pointer to supporting data for this error. An example would be a trace data set or dump data set.

- Subvector X'47'

  This subvector and its accompanying subfield are used as an internal focal point token for automatically correlating all alerts with the same token value. When requested, Netview internally searches the Netview database to display all alerts with the same token value.

  The value of subfield X'20' is in binary, and it should contain the *crrtkn* parameter followed by two bytes of binary zeros. The subfield contains *netid.luname.abcdef00* where *abcdef* is a 6-byte binary number and *00* are two bytes of binary zeros.

  This subvector and subfield are required in DRDA Level 2 if the *crrtkn* is available.

Alert Model BLKERR

This alert model is for blocking protocol error conditions discovered at the application requester. See the description for Table 11-3 on page 358 for a description of the subvectors, subfields, and code points.

Table 11-4 Alert Model BLKERR

Alert ID Number		X‘9A22708B’
Alert Type	X‘01’	Permanent
Alert Description	X‘2102’	Distributed Process Failed
Probable Causes	X‘1054’	Invalid Data Structure
User Causes	(none)	
Install Causes	(none)	
Failure Causes	X‘1054’ X‘1057’ X‘F0A3’	Invalid Data Structure Error Blocking Protocol Error Failure Occurred On (sf85)
Actions	X‘32D1’	Report The Following Logical Unit Of Work Identifier (sf85)(sf85)(sf85)
	X‘00B0’	Perform Problem Determination Procedure For (sf85)
	X‘00E1’	Perform (sf83) Problem Determination Procedures
	X‘0500’	Run Appropriate Trace
	X‘2203’	Review Supporting Data At Alert Sender
	X‘30E1’	Contact Service Representative For (sf83)
	X‘32D0’	Report The Following (sf85)(sf85)(sf85)
	X‘32A0’	Report The Following (sf85)
	...	...
Additional SVs	X‘10’ SV	Product Set Identifier
	X‘11’ SV	Product Identifier
	X‘08’ SF	Product Number
	X‘04’ SF	Version, Release, Modification
	X‘06’ SF	Product Common Name
	X‘05’ SV	Hierarchy / Resource List
	X‘10’ SF	Hierarchy Name List
	X‘11’ SF	Associated Resource List
	X‘01’ SV	Date / Time
	X‘10’ SF	Local Date / Time
	X‘48’ SV	Supporting Data Correlation
	X‘85’ SF	Detailed Data (Supporting data ptr)
	X‘47’ SV	MSU Correlation
	X‘20’ SF	CRRTKN (LUWID or UOWID format)
	...	...

Alert Model CHNVIO

This alert model is for Chaining Violation Error conditions discovered at the application requester. See the description for Table 11-3 on page 358 for a description of the subvectors, subfields, and code points.

Table 11-5 Alert Model CHNVIO

Alert ID Number		X'91EC5326'
Alert Type	X'01'	Permanent
Alert Description	X'2102'	Distributed Process Failed
Probable Causes	X'1054'	Invalid Data Structure
User Causes	(none)	
Install Causes	(none)	
Failure Causes	X'1054' X'1058' X'F0A3'	Invalid Data Structure Error Chaining Protocol Error Failure Occurred On (sf85)
Actions	X'32D1'	Report The Following Logical Unit Of Work Identifier (sf85)(sf85)(sf85)
	X'00B0'	Perform Problem Determination Procedure For (sf85)
	X'00E1'	Perform (sf83) Problem Determination Procedures
	X'0500'	Run Appropriate Trace
	X'2203'	Review Supporting Data At Alert Sender
	X'30E1'	Contact Service Representative For (sf83)
	X'32D0'	Report The Following (sf85)(sf85)(sf85)
	X'32A0'	Report The Following (sf85)
	...	...
Additional SVs	X'10' SV	Product Set Identifier
	X'11' SV	Product Identifier
	X'08' SF	Product Number
	X'04' SF	Version, Release, Modification
	X'06' SF	Product Common Name
	X'05' SV	Hierarchy/Resource List
	X'10' SF	Hierarchy Name List
	X'11' SF	Associated Resource List
	X'01' SV	Date/Time
	X'10' SF	Local Date/Time
	X'48' SV	Supporting Data Correlation
	X'85' SF	Detailed Data (Supporting data ptr)
	X'47' SV	MSU Correlation
	X'20' SF	CRRTKN (LUWID or UOWID format)
	...	...

Alert Model CMDCHK

This alert model is for Command Check conditions. See the description for Table 11-3 on page 358 for a description of the subvectors, subfields, and code points.

Table 11-6 Alert Model CMDCHK

Alert ID Number		X‘D67E885A’
Alert Type	X‘01’	Permanent
Alert Description	X‘2102’	Distributed Process Failed
Probable Causes	X‘1051’	Command Not Recognized
User Causes	(none)	
Install Causes	(none)	
Failure Causes	X‘1051’ X‘F0A3’	Command Not Recognized Failure Occurred On (sf85)
Actions	X‘32D1’	Report The Following Logical Unit Of Work Identifier (sf85)(sf85)(sf85)
	X‘00B0’	Perform Problem Determination Procedure For (sf85)
	X‘00E1’	Perform (sf83) Problem Determination Procedures
	X‘0500’	Run Appropriate Trace
	X‘2203’	Review Supporting Data At Alert Sender
	X‘30E1’	Contact Service Representative For (sf83)
	X‘32D0’	Report The Following (sf85)(sf85)(sf85)
	X‘32A0’	Report The Following (sf85)
	...	...
Additional SVs	X‘10’ SV	Product Set Identifier
	X‘11’ SV	Product Identifier
	X‘08’ SF	Product Number
	X‘04’ SF	Version, Release, Modification
	X‘06’ SF	Product Common Name
	X‘05’ SV	Hierarchy / Resource List
	X‘10’ SF	Hierarchy Name List
	X‘11’ SF	Associated Resource List
	X‘01’ SV	Date / Time
	X‘10’ SF	Local Date / Time
	X‘48’ SV	Supporting Data Correlation
	X‘85’ SF	Detailed Data (Supporting data ptr)
	X‘47’ SV	MSU Correlation
	X‘20’ SF	CRRTKN (LUWID or UOWID format)
	...	...

Alert Model CMDVLT

This alert model is for Command Violation conditions. This alert does not require support in DRDA Level 1. See the description for Table 11-3 on page 358 for a description of the subvectors, subfields, and code points.

Table 11-7 Alert Model CMDVLT

Alert ID Number		X‘4821F0B5’
Alert Type	X‘01’	Permanent
Alert Description	X‘2102’	Distributed Process Failed
Probable Causes	X‘1052’	Conversation Protocol
User Causes	(none)	
Install Causes	(none)	
Failure Causes	X‘109F’ X‘F0A3’	Command Violation Failure Occurred On (sf85)
Actions	X‘32D1’	Report The Following Logical Unit Of Work Identifier (sf85)(sf85)(sf85)
	X‘00B0’	Perform Problem Determination Procedure For (sf85)
	X‘00E1’	Perform (sf83) Problem Determination Procedures
	X‘0500’	Run Appropriate Trace
	X‘2203’	Review Supporting Data At Alert Sender
	X‘30E1’	Contact Service Representative For (sf83)
	X‘32D0’	Report The Following (sf85)(sf85)(sf85)
	X‘32A0’	Report The Following (sf85)
	...	...
Additional SVs	X‘10’ SV	Product Set Identifier
	X‘11’ SV	Product Identifier
	X‘08’ SF	Product Number
	X‘04’ SF	Version, Release, Modification
	X‘06’ SF	Product Common Name
	X‘05’ SV	Hierarchy / Resource List
	X‘10’ SF	Hierarchy Name List
	X‘11’ SF	Associated Resource List
	X‘01’ SV	Date / Time
	X‘10’ SF	Local Date / Time
	X‘48’ SV	Supporting Data Correlation
	X‘85’ SF	Detailed Data (Supporting data ptr)
	X‘47’ SV	MSU Correlation
	X‘20’ SF	CRRTKN (LUWID or UOWID format)
	...	...

Alert Model DSCERR

This alert model is for the data descriptor error conditions. See the description for Table 11-3 on page 358for a description of the subvectors, subfields, and code points.

Table 11-8 Alert Model DSCERR

Alert ID Number		X‘2257C33F’
Alert Type	X‘01’	Permanent
Alert Description	X‘2102’	Distributed Process Failed
Probable Causes	X‘1053’	Data Descriptor
User Causes	(none)	
Install Causes	(none)	
Failure Causes	X‘1053’ X‘F0A3’	Data Descriptor Error Failure Occurred On (sf85)
Actions	X‘32D1’	Report The Following Logical Unit Of Work Identifier (sf85)(sf85)(sf85)
	X‘00B0’	Perform Problem Determination Procedure For (sf85)
	X‘00E1’	Perform (sf83) Problem Determination Procedures
	X‘0500’	Run Appropriate Trace
	X‘2203’	Review Supporting Data At Alert Sender
	X‘30E1’	Contact Service Representative For (sf83)
	X‘32D0’	Report The Following (sf85)(sf85)(sf85)
	X‘32A0’	Report The Following (sf85)
	...	...
Additional SVs	X‘10’ SV	Product Set Identifier
	X‘11’ SV	Product Identifier
	X‘08’ SF	Product Number
	X‘04’ SF	Version, Release, Modification
	X‘06’ SF	Product Common Name
	X‘05’ SV	Hierarchy / Resource List
	X‘10’ SF	Hierarchy Name List
	X‘11’ SF	Associated Resource List
	X‘01’ SV	Date / Time
	X‘10’ SF	Local Date / Time
	X‘48’ SV	Supporting Data Correlation
	X‘85’ SF	Detailed Data (Supporting data ptr)
	X‘47’ SV	MSU Correlation
	X‘20’ SF	CRRTKN (LUWID or UOWID format)
	...	...

Alert Model GENERR

This alert model is for error conditions that need an alert, but do not have a more specific alert model to choose from. See the description for Table 11-3 on page 358 for a description of the subvectors, subfields, and code points.

Table 11-9 Alert Model GENERR

Alert ID Number		X'46E34E31'
Alert Type	X'12'	Unknown
Alert Description	X'2102'	Distributed Process Failed
Probable Causes	X'1000'	Software Program:
User Causes	(none)	
Install Causes	(none)	
Failure Causes	X'10E1' X'F0A3'	Software Program (sf83) Failure Occurred On (sf85)
Actions	X'32D1'	Report The Following Logical Unit Of Work Identifier (sf85)(sf85)(sf85)
	X'00B0'	Perform Problem Determination Procedure For (sf85)
	X'00E1'	Perform (sf83) Problem Determination Procedures
	X'0500'	Run Appropriate Trace
	X'2203'	Review Supporting Data At Alert Sender
	X'30E1'	Contact Service Representative For (sf83)
	X'32D0'	Report The Following (sf85)(sf85)(sf85)
	X'32A0'	Report The Following (sf85)
	...	...
Additional SVs	X'10' SV	Product Set Identifier
	X'11' SV	Product Identifier
	X'08' SF	Product Number
	X'04' SF	Version, Release, Modification
	X'06' SF	Product Common Name
	X'05' SV	Hierarchy/Resource List
	X'10' SF	Hierarchy Name List
	X'11' SF	Associated Resource List
	X'01' SV	Date/Time
	X'10' SF	Local Date/Time
	X'48' SV	Supporting Data Correlation
	X'85' SF	Detailed Data (Supporting data ptr)
	X'47' SV	MSU Correlation
	X'20' SF	CRRTKN (LUWID or UOWID format)
	...	...

Alert Model PRCCNV

This alert model is for the Conversation Protocol Error condition. See the description for Table 11-3 on page 358 for a description of the subvectors, subfields, and code points.

Table 11-10 Alert Model PRCCNV

Alert ID Number		X'DA23E856'
Alert Type	X'01'	Permanent
Alert Description	X'2102'	Distributed Process Failed
Probable Causes	X'1052'	Conversation Protocol
User Causes	(none)	
Install Causes	(none)	
Failure Causes	X'1052' X'F0A3'	Conversation Protocol Error Failure Occurred On (sf85)
Actions	X'32D1'	Report The Following Logical Unit Of Work Identifier (sf85)(sf85)(sf85)
	X'00B0'	Perform Problem Determination Procedure For (sf85)
	X'00E1'	Perform (sf83) Problem Determination Procedures
	X'0500'	Run Appropriate Trace
	X'2203'	Review Supporting Data At Alert Sender
	X'30E1'	Contact Service Representative For (sf83)
	X'32D0'	Report The Following (sf85)(sf85)(sf85)
	X'32A0'	Report The Following (sf85)
	...	...
Additional SVs	X'10' SV	Product Set Identifier
	X'11' SV	Product Identifier
	X'08' SF	Product Number
	X'04' SF	Version, Release, Modification
	X'06' SF	Product Common Name
	X'05' SV	Hierarchy / Resource List
	X'10' SF	Hierarchy Name List
	X'11' SF	Associated Resource List
	X'01' SV	Date / Time
	X'10' SF	Local Date / Time
	X'48' SV	Supporting Data Correlation
	X'85' SF	Detailed Data (Supporting data ptr)
	X'47' SV	MSU Correlation
	X'20' SF	CRRTKN (LUWID or UOWID format)
	...	...

Alert Model QRYERR

This alert model is for Cursor Error conditions. See the description for Table 11-3 on page 358for a description of the subvectors, subfields, and code points.

Table 11-11 Alert Model QRYERR

Alert ID Number		X‘3AED0327’
Alert Type	X‘01’	Permanent
Alert Description	X‘2102’	Distributed Process Failed
Probable Causes	X‘1055’	Invalid Cursor State
User Causes	(none)	
Install Causes	(none)	
Failure Causes	X‘1055’ X‘F0A3’	Invalid Cursor State Failure Occurred On (sf85)
Actions	X‘32D1’	Report The Following Logical Unit Of Work Identifier (sf85)(sf85)(sf85)
	X‘00B0’	Perform Problem Determination Procedure For (sf85)
	X‘00E1’	Perform (sf83) Problem Determination Procedures
	X‘0500’	Run Appropriate Trace
	X‘2203’	Review Supporting Data At Alert Sender
	X‘30E1’	Contact Service Representative For (sf83)
	X‘32D0’	Report The Following (sf85)(sf85)(sf85)
	X‘32A0’	Report The Following (sf85)
	...	...
Additional SVs	X‘10’ SV	Product Set Identifier
	X‘11’ SV	Product Identifier
	X‘08’ SF	Product Number
	X‘04’ SF	Version, Release, Modification
	X‘06’ SF	Product Common Name
	X‘05’ SV	Hierarchy/Resource List
	X‘10’ SF	Hierarchy Name List
	X‘11’ SF	Associated Resource List
	X‘01’ SV	Date/Time
	X‘10’ SF	Local Date/Time
	X‘48’ SV	Supporting Data Correlation
	X‘85’ SF	Detailed Data (Supporting data ptr)
	X‘47’ SV	MSU Correlation
	X‘20’ SF	CRRTKN (LUWID or UOWID format)
	...	...

Alert Model RDBERR

This alert model is for Relational Database access errors. See the description for Table 11-3 on page 358for a description of the subvectors, subfields, and code points.

Table 11-12 Alert Model RDBERR

Alert ID Number		X‘36B0632B’
Alert Type	X‘01’	Permanent
Alert Description	X‘2102’	Distributed Process Failed
Probable Causes	X‘1056’	Relational Database Access
User Causes	(none)	
Install Causes	(none)	
Failure Causes	X‘1056’ X‘F0A3’	Relational Database Access Error Failure Occurred On (sf85)
Actions	X‘32D1’	Report The Following Logical Unit Of Work Identifier (sf85)(sf85)(sf85)
	X‘00B0’	Perform Problem Determination Procedure For (sf85)
	X‘00E1’	Perform (sf83) Problem Determination Procedures
	X‘0500’	Run Appropriate Trace
	X‘2203’	Review Supporting Data At Alert Sender
	X‘30E1’	Contact Service Representative For (sf83)
	X‘32D0’	Report The Following (sf85)(sf85)(sf85)
	X‘32A0’	Report The Following (sf85)
	...	...
Additional SVs	X‘10’ SV	Product Set Identifier
	X‘11’ SV	Product Identifier
	X‘08’ SF	Product Number
	X‘04’ SF	Version, Release, Modification
	X‘06’ SF	Product Common Name
	X‘05’ SV	Hierarchy / Resource List
	X‘10’ SF	Hierarchy Name List
	X‘11’ SF	Associated Resource List
	X‘01’ SV	Date / Time
	X‘10’ SF	Local Date / Time
	X‘48’ SV	Supporting Data Correlation
	X‘85’ SF	Detailed Data (Supporting data ptr)
	X‘47’ SV	MSU Correlation
	X‘20’ SF	CRRTKN (LUWID or UOWID format)
	...	...

Alert Model RSCLMT

This alert model is for the Resource Limit Reached condition. See the description for Table 11-3 on page 358 for a description of the subvectors, subfields, and code points.

Table 11-13 Alert Model RSCLMT

Alert ID Number		X‘A70F6F9E’
Alert Type	X‘01’	Permanent
Alert Description	X‘2102’	Distributed Process Failed
Probable Causes	X‘1057’	Resource Limit Reached
User Causes	(none)	
Install Causes	(none)	
Failure Causes	X‘F0C0’ X‘F0A3’	Resource Limit Reached (sf85)(sf85) Failure Occurred On (sf85)
Actions	X‘32D1’	Report The Following Logical Unit Of Work Identifier (sf85)(sf85)(sf85)
	X‘00B0’	Perform Problem Determination Procedure For (sf85)
	X‘00E1’	Perform (sf83) Problem Determination Procedures
	X‘0500’	Run Appropriate Trace
	X‘2203’	Review Supporting Data At Alert Sender
	X‘30E1’	Contact Service Representative For (sf83)
	X‘32D0’	Report The Following (sf85)(sf85)(sf85)
	X‘32A0’	Report The Following (sf85)
	...	...
Additional SVs	X‘10’ SV	Product Set Identifier
	X‘11’ SV	Product Identifier
	X‘08’ SF	Product Number
	X‘04’ SF	Version, Release, Modification
	X‘06’ SF	Product Common Name
	X‘05’ SV	Hierarchy / Resource List
	X‘10’ SF	Hierarchy Name List
	X‘11’ SF	Associated Resource List
	X‘01’ SV	Date / Time
	X‘10’ SF	Local Date / Time
	X‘48’ SV	Supporting Data Correlation
	X‘85’ SF	Detailed Data (Supporting data ptr)
	X‘47’ SV	MSU Correlation
	X‘20’ SF	CRRTKN (LUWID or UOWID format)
	...	...

The two subfield X‘85’s for the Failure Cause code point X‘F0C0’ should contain the following data beginning at byte 5 (see Figure 11-5 on page 372). The subfield X‘85’s are shown below in the order they should appear in the Failure Causes Subfield. The first subfield X‘85’ uses the data ID code point X‘00A7’ for resource. The detailed data field contains the name of the resource that has reached a limit, if available. If the resource name is not available, then the DDM code point for the resource type should be used. The second subfield X‘85’ uses the data ID code point X‘000E’ for reason code. The detailed data field contains the product-dependent reason code for this error. If the reason code is not available, then this subfield uses the data ID code point of X‘0000’ and does not contain any data.

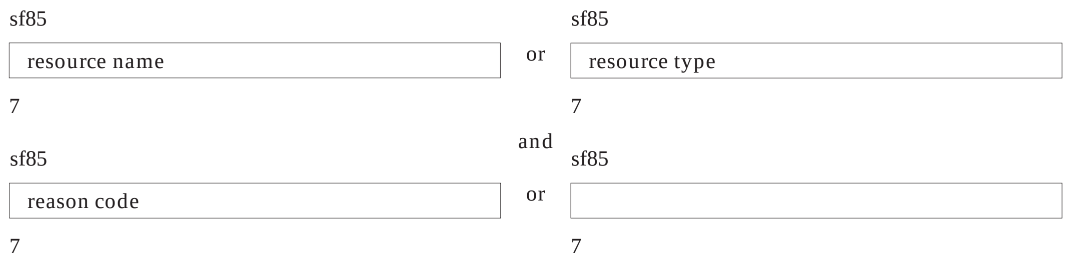


Figure 11-5 Subfield X‘85’s for Failure Causes Code Point X‘F0C0’

Alert Model SECVIOL

This alert model is for security violation error conditions discovered at the application requester or application server. See the description for Table 11-3 on page 358for a description of the subvectors, subfields, and code points.

Table 11-14 Alert Model SECVIOL

Alert ID Number		X‘50C0C0BC’
Alert Type	X‘01’	Permanent
Alert Description	X‘2102’	Distributed Process Failed
Probable Causes	X‘6700’	Security Problem
User Causes	(none)	
Install Causes	(none)	
Failure Causes	X‘107F’ X‘F0A3’	Distribution Session Not Created Failure Occurred On (sf85)
Actions	X‘32D1’	Report The Following Logical Unit Of Work Identifier (sf85)(sf85)(sf85)
	X‘00B0’	Perform Problem Determination Procedure For (sf85)
	X‘00E1’	Perform (sf83) Problem Determination Procedures
	X‘0500’	Run Appropriate Trace
	X‘2203’	Review Supporting Data At Alert Sender
	X‘30E1’	Contact Service Representative For (sf83)
	X‘32D0’	Report The Following (sf85)(sf85)(sf85)
	X‘32A0’	Report The Following (sf85)
	...	...
Additional SVs	X‘10’ SV	Product Set Identifier
	X‘11’ SV	Product Identifier
	X‘08’ SF	Product Number
	X‘04’ SF	Version, Release, Modification
	X‘06’ SF	Product Common Name
	X‘05’ SV	Hierarchy/Resource List
	X‘10’ SF	Hierarchy Name List
	X‘11’ SF	Associated Resource List
	X‘01’ SV	Date/Time
	X‘10’ SF	Local Date/Time
	X‘48’ SV	Supporting Data Correlation
	X‘85’ SF	Detailed Data (Supporting data ptr)
	X‘47’ SV	MSU Correlation
	X‘20’ SF	CRRTKN (LUWID or UOWID format)
	...	...

Alert Model SYNTAX

This alert model is for the Data Stream Syntax Error condition. See the description for Table 11-3 on page 358for a description of the subvectors, subfields, and code points.

Table 11-15 Alert Model SYNTAX

Alert ID Number		X'C299284E'
Alert Type	X'01'	Permanent
Alert Description	X'2102'	Distributed Process Failed
Probable Causes	X'1054'	Invalid Data Structure
User Causes	(none)	
Install Causes	(none)	
Failure Causes	X'1054' X'F0A3'	Invalid Data Structure Error Failure Occurred On (sf85)
Actions	X'32D1'	Report The Following Logical Unit Of Work Identifier (sf85)(sf85)(sf85)
	X'00B0'	Perform Problem Determination Procedure For (sf85)
	X'00E1'	Perform (sf83) Problem Determination Procedures
	X'0500'	Run Appropriate Trace
	X'2203'	Review Supporting Data At Alert Sender
	X'30E1'	Contact Service Representative For (sf83)
	X'32D0'	Report The Following (sf85)(sf85)(sf85)
	X'32A0'	Report The Following (sf85)
	...	...
Additional SVs	X'10' SV	Product Set Identifier
	X'11' SV	Product Identifier
	X'08' SF	Product Number
	X'04' SF	Version, Release, Modification
	X'06' SF	Product Common Name
	X'05' SV	Hierarchy / Resource List
	X'10' SF	Hierarchy Name List
	X'11' SF	Associated Resource List
	X'01' SV	Date / Time
	X'10' SF	Local Date / Time
	X'48' SV	Supporting Data Correlation
	X'85' SF	Detailed Data (Supporting data ptr)
	X'47' SV	MSU Correlation
	X'20' SF	CRRTKN (LUWID or UOWID format)
	...	...

11.3.4 Alert Example

This section provides an alert example.

11.3.4.1 Major Vector/Subvector/Subfield Construction

Figure 11-6 is a graphical representation of the major vector for an alert. It is comprised of a length field, a key field, and multiple subvectors. The subvectors are comprised of other subvectors and subfields. The subfields are comprised of data.

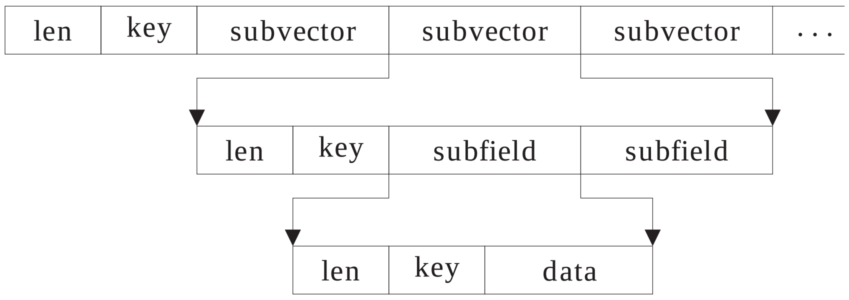


Figure 11-6 Major Vector / Subvector / Subfield Construction

Figure 11-7 on page 376 is an example of an alert for an AGNPRMRM with severity code of 64. This figure shows the Alert Major Vector that would be passed to a focal point. The related subfields for each subvector are grouped together and labeled for ease of reading. The hexadecimal identifiers for the fields serve two purposes. They are the actual hexadecimal offsets into the major vector, and they are identifiers for the descriptions of these fields. The descriptions of the fields follow the figure.

If two Product Set ID (X‘10’) subvectors are present, the first one is interpreted as the Alert sender and the second one is interpreted as the resource experiencing the problem. If the resource experiencing the problem is the resource sending the alert, then only one Product Set ID subvector should be present.

Alert Major Vector

[0]	[2]
00EA	0000

Hierarchy / Resource List Subvector and Accompanying Hierarchy Name List Subfield and Associated Resource List Subfield

[4]	[5]
38	05

[6]	[7]	[8]	[9]	[A]	[C]	[D]
0D	10	00	03	AR	40	42

[E]	[F]	[11]	[12]
03	AS	00	43

[13]	[14]	[15]	[16]	[17]	[1F]	[20]
29	11	00	09	AR APPL1	00	40

[21]	[22]	[2D]	[2E]
0C	AR sscpname	00	F4

[2F]	[30]	[3A]	[3B]
0B	AS rdbname	00	41

Generic Alert Data Subvector

[3C]	[3D]	[3E]	[40]	[41]	[43]
0B	92	0000	01	2102	2E0AA333

Figure 11-7 Alert Example for AGNPRMRM with Severity Code of 64 (Part 1)

Probable Causes Subvector

[47]	[48]	[49]	
04	93	1050	

Failure Causes Subvector and Accompanying Failure Causes Subfield, Detailed Data Subfields and Recommended Actions Subfield

[4B]	[4C]	
5C	96	

[4D]	[4E]	[4F]	
04	01	1050	

[51]	[52]	[53]	
19	01	F0A3	

[55]	[56]	[57]	[58]	[59]	[5B]	[5C]	
17	85	90	xx	0087	11	CCEEEErestofname	

[6C]	[6D]	[6E]	
36	81	32D1	

[70]	[71]	[72]	[73]	[74]	[76]	[77]	
19	85	90	xx	0000	11	netidddd.lunameee.	

[89]	[8A]	[8B]	[8C]	[8D]	[8F]	[90]	
10	85	90	xx	0000	11	BA9876543210.0001	

[A1]	[A2]	
02	85	

[A3]	[A4]	[A5]	
04	81	2203	

Figure 11-8 Alert Example for AGNPRMRM with Severity Code of 64 (Part 2)

Product Set ID Subvector and Accompanying Product ID Subvector, Software Program Number Subfield, Software Product Common Name Subfield, and Software Product Common Level Subfield

[A7]	[A8]	[A9]
23	10	r

[AA]	[AB]	[AC]
20	11	04

[AD]	[AE]	[AF]
09	08	5665DB2

[B6]	[B7]	[B8]
0C	06	DATABASE 2*

[C2]	[C3]	[C4]	[C6]	[C8]
08	04	02	03	00

Date / Time Subvector and Accompanying Local Date / Time Subfield

[CA]	[CB]	[CC]
0D	01	10

[CD]	[CE]	[CF]	[D0]	[D1]	[D2]	[D3]	[D4]	[D5]
0A	10	58	0C	02	0F	0E	05	0FA3

Supporting Data Subvector and Accompanying Detailed Data Subfield

[D7]	[D8]
12	48

[D9]	[DA]	[DB]	[DC]	[DD]	[DF]	[E0]
12	85	90	xx	00DA	11	SYS1.LOGREC

Figure 11-9 Alert Example for AGNPRMRM with Severity Code of 64 (Part 3)

These are the descriptions of the fields in the alert example:

0 Length of MS Major Vector (2 bytes).

2 Key for MS Major Vector (2 bytes).

4 Length of Hierarchy Resource List Subvector in binary (1 byte).

5	Key for Hierarchy Resource List Subvector (1 byte).
6	Length of Hierarchy Name List Subfield in binary (1 byte).
7	Key for Hierarchy Name List Subfield (1 byte).
8	Bit 0=0 so alert receiver does not modify the list. Bits 1 through 7 are reserved (1 byte).
9	Length of resource name + 1 in binary (1 byte).
A	Resource name in uppercase alphanumeric EBCDIC characters. The name must not exceed 8 characters. The name in the example is application requester for application requester (2 bytes).
C	Bit 0 is reserved. Bit 1 is equal to 0 or 1 dependent on whether this resource should be displayed if the alert receiver can only display one resource. In the example, application requester would not be displayed. Bits 2 through 7 are reserved (1 byte).
D	Resource type identifier (1 byte).
E	Length of resource name + 1 in binary (1 byte).
F	Resource name. The name in the example is application server for the application server. See field 9 for more information (2 bytes).
11	Bit 0 is reserved. Bit 1 is equal to 0 or 1 dependent on whether this resource should be displayed if the alert receiver can only display one resource. In the example, application server would be displayed. Bits 2 through 7 are reserved (1 byte).
12	Resource type identifier (1 byte).
13	Length of Associated Resources Subfield in binary (1 byte).
14	Key for Associated Resources Subfield (1 byte).
15	Reserved (1 byte).
16	Length of resource name + 1 in binary (1 byte).
17	The name of the resource in uppercase alphanumeric EBCDIC characters. The resource with which it is associated precedes the name. This field is not to exceed 56 characters (8 bytes). In the example the name of the resource is APPL1 and is associated with the resource application requester (8 bytes).
1F	Flags (1 byte).
20	Resource type identifier (1 byte).
21-3B	Two more associated resource entries and they follow the same format as fields 16-20 (27 bytes).
3C	Length of Generic Alert Data Subvector in binary (1 byte).
3D	Key for Generic Alert Data Subvector (1 byte).
3E	Bits 0, 1, and 2 equal 0 to represent alert is not directly initiated by operator, alert was sent when the problem was detected, and alert sender is not reporting a previously detected alert condition. Bits 3 through 15 are reserved (2 bytes).
40	Alert type, Permanent Error (1 byte).
41	Alert description code (2 bytes).
43	Alert ID number (4 byte hexadecimal value).

47	Length of Probable Causes Subvector in binary (1 byte).
48	Key for Probable Causes Subvector (1 byte).
49	Probable causes code points (2 bytes).
4B	Length of Failure Causes Subvector in binary (1 byte).
4C	Key for Failure Causes Subvector (1 byte).
4D	Length of Failure Causes Subfield in binary (1 byte).
4E	Key for Failure Causes Subfield (1 byte).
4F	Failure Causes code point (2 bytes).
51	Length of Failure Causes Subfield in binary (1 byte).
52	Key for Failure Causes Subfield (1 byte).
53	Failure Causes code point (2 bytes).
55	Length of Detailed Data Subfield in binary (1 byte).
56	Key for Detailed Data Subfield (1 byte).
57	Product ID code. Bits 0 through 3 equal 9 to indicate the product ID subvector being indexed and the particular data to be extracted from this subvector. In this example, it is a software product common name. Bit 4=0 for the alert sender Product Set ID. Bits 5 through 7 equal 0 to indicate the first Product Set ID subvector of the type defined above, should be used (1 byte).
58	Reserved (1 byte).
59	Data ID equals 0087 (2 bytes).
5B	Data encoding equals 11 for character set 00640-0500 (1 byte).
5C	Detailed data. The example shows the RDB_NAME spelled out (16 bytes).
6C	Length of Recommended Actions Subfield in binary (1 byte).
6D	Key for Recommended Actions Subfield (1 byte).
6E	Recommended action code point indicating Report The Following Logical Unit Of Work Identifier (2 bytes).
70-77	The first subfield X‘85’. It has the netid.luname portion of the LUWID or UOWID (25 bytes).
89-90	The second subfield X‘85’. It has the instance and sequence number portions of the LUWID or UOWID. The data displayed is the character representation of the 6-byte binary instance number, followed by a period and the character representation of the 2-byte binary sequence number (24 bytes).
A1-A2	The last subfield X‘85’. It is blank (2 bytes).
A3	Length of Recommended Actions Subfield in binary (1 byte).
A4	Key for Recommended Actions Subfield (1 byte).
A5	Recommended action code point indicating Review Supporting Data at Alert Sender (2 bytes).
A7	Length of Product Set ID Subvector in binary (1 byte).

A8	Key for Product Set ID Subvector (1 byte).
A9	Retired (1 byte).
AA	Length of Product Identifier Subvector in binary (1 byte).
AB	Key for Product Identifier Subvector (1 byte).
AC	Bits 0-3 are reserved. Bits 4-7 equal 4 to indicate the software level (1 byte).
AD	Length of Product Identifier Subfield in binary (1 byte).
AE	Key for Product Identifier Subfield (1 byte).
AF	Software product program number. Seven uppercase alphanumeric EBCDIC characters (7 bytes).
B6-C8	These fields are two more Product Identifier subfields. The first one is Software Product Common Name and the second is Software Product Common Level with version, release, and modification level (20 bytes).
CA	Length of Date / Time Subvector in binary (1 byte).
CB	Key for Date / Time Subvector (1 byte).
CC	Indicates the Date / Time is the local Date / Time (1 byte).
CD	Length of Local Date / Time Subfield in binary (1 byte).
CE	Key for Local Date / Time Subfield (1 byte).
CF-D1	The year, month, and day in binary (3 bytes).
D2-D4	The hours, minutes, and seconds in binary (3 bytes).
D5	The extension of time in binary and provides fractions of seconds (2 bytes).
D7	Length of Supporting Data Correlation Subvector in binary (1 byte).
D8	Key for Supporting Data Correlation Subvector (1 byte).
D9	Length of Detailed Data Subfield in binary (1 byte).
DA	Key for Detailed Data Subfield (1 byte).
DB	Product ID code (1 byte).
DC	Reserved (1 byte).
DD	Data ID equals X‘00DA’ for Log ID (2 bytes).
DF	Data encoding equals 11 for character set 00640-0500 (1 byte).
E0	Detailed data. The example shows Sys1.Logrec as the log ID (11 bytes).

Open Group Technical Standard

Part 3:

Network Protocols

The Open Group

Chapter 12

SNA

This chapter summarizes the characteristics of DRDA communications flow using the SNA network environment.

12.1 SNA and the DDM Communications Model

SNA implementations of DRDA use the DDM Communications Managers. The DDM LU 6.2 Conversational Communications Manager (CMNAPPC) supports the base and option set functions of LU 6.2 required by DRDA Level 1 implementations and DRDA Level 2 or DRDA Level 3 implementations without resource recovery support. The DDM LU 6.2 Sync Point Conversational Communications Manager (CMNSYNCPT) supports the base and option set functions, including synchronization point support, that distributed unit of work implementations require for coordinated resource recovery support. For further detail, see the DDM terms CMNAPPC and CMNSYNCPT in the DDM Reference.

12.2 What You Need to Know About SNA and LU 6.2

This chapter assumes some familiarity with Systems Network Architecture (SNA) concepts and with LU 6.2. With a general exposure to these topics, it should be possible to understand how DRDA's use of LU 6.2 function compares with the many other types of usage that the general-purpose LU 6.2 architecture permits. With more detailed knowledge, it should be possible to understand how to use LU 6.2 function in DRDA environments. For a list of relevant LU 6.2 publications, see **Referenced Documents** on page xxiv.

The reader should also have some familiarity with DDM terms and the DDM model. A reader with a general exposure to DDM should be able to understand how DRDA's use of LU 6.2 relates to the DDM communications managers of the DDM model.

Refer to **Referenced Documents** on page xxiv for the list of DDM publications.

12.3 LU 6.2

Logical Unit type 6.2 (LU 6.2) is the architecture for advanced program-to-program communication (APPC). Products that implement LU 6.2 provide program-to-program communications that are robust enough for distributed database management processing. The robust features necessary for distributed database include:

- Timely failure notification

  LU 6.2 is the program-to-program architecture that guarantees timely notification of network connection and end node failures. Knowing when one user is done is of the utmost importance in a database management environment where potentially thousands of users can be sharing information.

- Propagation of security, authentication, authorization, and accounting information

  LU 6.2 provides and permits the propagation of who, what, when, and where information among the resource managers participating in a user transaction. Security, authentication, and authorization information is essential for the proper control of access to managed data. Accounting information is essential for the tracking of resource use and consumption.

- Synchronization point support

  LU 6.2 provides support for coordinating updates across multiple systems. This is done through resource recovery processing, which includes two-phase commit protocols on LU 6.2 conversations. This feature is not supported in DRDA Level 1.

DRDA relies on a subset of the LU 6.2 defined function. That subset includes function provided by verbs from both the LU 6.2 base and option sets. This chapter identifies the LU 6.2 function contained within the DRDA subset, relates the DRDA subset to DDM terms and the DDM model, and discusses the characteristics of DRDA communications flows that are unique to DRDA.

12.4 LU 6.2 Verb Categories

The LU 6.2 protocol boundary consists of two categories of verbs: conversation verbs and control-operator verbs.

1. Conversation verbs define the means for program-to-program communication. The three types of conversation verbs are mapped, basic, and type-independent.

   - Mapped conversation verbs provide functions for application programs written in high-level application program languages. Application transaction programs use mapped conversation verbs.
   - Basic conversation verbs provide functions for end-user services or protocol boundaries for end-user application transaction programs. LU services programs use basic conversation verbs.
   - Type-independent verbs provide functions that span both mapped and basic conversation types (such as synchronization point services). Both application transaction programs and LU services programs use type-independent verbs.

2. Control-operator verbs define the means for program or operator control of the LU's resources. Control-operator transaction programs use control-operator verbs to assist the control operator in performing functions related to the control of an LU. LU 6.2 implementations that employ parallel sessions use control-operator verbs to define the parallel session support that is available between them.

12.5 LU 6.2 Product-Support Subsetting

LU 6.2 product-support subsetting of the verbs is defined by means of function groups or sets. A set consists of all the functions that together represent an indivisible group for products to implement; that is, a product implementing a particular set implements all of the functions within the set.

The base set is the set of LU 6.2 verbs, parameters, return codes, and what-received indications that all programmable LU 6.2 products support.

The option sets are the sets of LU 6.2 verbs, parameters, return codes, and what-received indications that a product can support depending on the product. A product can support any number of options sets or none. If a product supports an option set, then the product must support all verbs, parameters, return codes, and what-received indications defined in the option set.

12.6 LU 6.2 Base and Option Sets

Implementations of DRDA must use LU 6.2 for communications and in support of security, accounting, and transaction processing. Due to the complexity of distributed database management system processing, DRDA requires both base and option set functions of LU 6.2.

Application requesters (ARs) and application servers (ASs) use basic conversation verbs. Unless otherwise noted, all application requesters and application servers use each LU 6.2 function and must accomplish their goals using the verbs listed below or equivalent local interfaces.

Any verbs outside the set listed in DRDA are not required by DRDA, and DRDA does not provide any architecture for use of those verbs.

12.6.1 Base Set Functions

DRDA requires base set functions from the basic conversation and type-independent verb categories.

12.6.1.1 Basic Conversation Verb Category

DRDA uses base set function provided by the following basic conversation verbs:

- ALLOCATE
- DEALLOCATE
- GET_ATTRIBUTES
- RECEIVE_AND_WAIT
- SEND_DATA
- SEND_ERROR

12.6.1.2 Type-Independent Verb Category

DRDA uses base set function provided by the following type-independent conversation verb:

- GET_TP_PROPERTIES

12.6.2 Option Set Functions

DRDA requires option set functions from the basic conversation verb category and type-independent verb category. The numbers in the parentheses are option set numbers. If a verb does not have an option set number, the verb is in the base set, but the function or variable included to perform the function is an option set function. See the *SNA Transaction Programmer's Reference Manual for LU Type 6.2* (GC30-3084, IBM) for details about option set numbers.

12.6.2.1 Basic Conversation Verb Category

DRDA uses option set function provided by the following basic conversation verbs:

- User ID Verification (Conversation-Level Security) (212)[46]

---

46. LU 6.2 Conversation-Level Security is optional if DCE user authentication mechanisms are in use.

ALLOCATE

- Program-Supplied User ID and Password (Conversation-Level Security) (213)[47]

  ALLOCATE

- Specify a synchronization level of SYNCPT (108)[48]

  ALLOCATE

- Get the conversation state (108)[49]

  GET_ATTRIBUTES

- PREPARE_TO_RECEIVE (105)

  Only application requesters or application servers that require asynchronous receive capabilities need use PREPARE_TO_RECEIVE.

- POST_ON_RECEIPT with TEST for Posting (103)

  Only application requesters or application servers that require asynchronous receive capabilities need use POST_ON_RECEIPT with TEST for Posting.

  - POST_ON_RECEIPT
  - TEST

12.6.2.2 Type-Independent Verb Category

DRDA uses option set function provided by the following type-independent conversation verbs:

- LUW_Identifier (243)

  GET_TP_PROPERTIES

- Protected_LUW_Identifier (108)[50]

  GET_TP_PROPERTIES

- SYNCPT (108)[51]
- BACKOUT (108)[52]
- SET_SYNCPT_OPTIONS (108)[53]

---

47. LU 6.2 Conversation-Level Security is optional if DCE user authentication mechanisms are in use.
48. Not supported in DRDA Level 1.
49. Not supported in DRDA Level 1.
50. Not supported in DRDA Level 1.
51. Not supported in DRDA Level 1.

    Syncpt and Backout are the LU 6.2 verbs and terms for committing and rolling back the work, respectively. Because Commit and Rollback are the accepted terms in relational databases to perform the function of committing and rolling back the work, this reference will use the terms commit and rollback wherever the context is not directly related to LU 6.2.

52. Not supported in DRDA Level 1.

    Syncpt and Backout are the LU 6.2 verbs and terms for committing and rolling back the work, respectively. Because Commit and Rollback are the accepted terms in relational databases to perform the function of committing and rolling back the work, this reference will use the terms commit and rollback wherever the context is not directly related to LU 6.2.

53. Not supported in DRDA Level 1.

    SET_SYNCPT_OPTIONS is a verb in support of LU 6.2 verbs that provides synchronization point optimizations. DRDA encourages the implementation of the synchronization point optimizations, but does not rely on or require the implementation of these optimizations. If an implementation chooses to implement the optimization that allows a resource to vote read-only during resource recovery processing, the resource cannot vote read-only if there are held cursors at that resource.

12.7 LU 6.2 and DRDA

Application requesters and application servers that provide DRDA capabilities use DRDA flows. DRDA flows permit implementations of DRDA to initialize conversations, terminate conversations, and process DRDA requests.

12.7.1 Initializing a Conversation

Initialization processing allocates a conversation and prepares a DRDA execution environment. Only an application requester can start a conversation. Authentication occurs during initialization processing through the required use of Conversation-Level Security (end-user verification) as specified in the LU 6.2 architecture. The use of conversation-level security verifies the end-user name associated with the conversation. Database management systems verify that authenticated IDs have the authorization to perform DRDA database manager requests.

Refer to Section 6.1 on page 270 and Section 6.1.1 on page 270for a detailed description of architected end-user names.

Authentication between an application requester and application server occurs once per conversation during ALLOCATE processing.

Initialization processing propagates the resource recovery level that is required for a particular conversation. This is carried in the SYNC_LEVEL parameter of the LU 6.2 ALLOCATE verb.

Initialization processing also propagates basic accounting information. An LU 6.2 ALLOCATE verb within the initialization flow specifies an end-user name, a logical unit of work ID (LUWID), remote LUNAME, and transaction program name to provide the who, what, when, and where information useful for accounting in DRDA environments.

The DDM Reference provides a general overview of the component communications flows that comprise a DRDA initialization flow. See the DDM terms APPCMNI and SYNCMNI, which discuss initiation of LU 6.2 communications.

12.7.1.1 LU 6.2 Verbs that the Application Requester Uses

The LU 6.2 verbs that the application requester uses for DRDA initialization flows are described here. Unless otherwise specified, refer to the *SNA Transaction Programmer's Reference Manual for LU Type 6.2* (GC30-3084, IBM) and to the *SNA LU 6.2 Reference: Peer Protocols* (SC31-6808, IBM) for further detail.

ALLOCATE

ALLOCATE initiates a requester initialization verb sequence. The execution of the verb first ensures that a session exists between the LU of an application requester and a remote LU, and then allocates a basic conversation on that session between the application requester and the specified remote transaction program (TP).

The LU_NAME value is a fully qualified LUNAME, as specified in the LU 6.2 architecture. The LU 6.2 architecture requires the LU_NAME parameter and continues to permit use of unqualified LU_NAME values only for migration purposes. Products that do not support fully qualified LU_NAME values can have difficulties working in SNA network interconnect environments.

The transaction program name value can be a registered DRDA transaction program name, registered DDM transaction program name, or any non-registered transaction program name. Refer to Section 6.8 on page 279 for further detail.

Applications using the SQL language are not required to understand LU_NAME values (qualified or unqualified) nor transaction program name values. The external name that an application can use is RDB_NAME. DRDA does not define the mechanism by which the application requester derives the NETID.LU_NAME and transaction program name pair from the RDB_NAME. DRDA permits the association of multiple RDB_NAMEs with a single transaction program name and NETID_LUNAME.

The TYPE parameter value must be BASIC_CONVERSATION. DRDA has no usage requirement for mapped conversations.

The SYNC_LEVEL parameter value must be NONE for DRDA Level 1 and can be SYNCPT for DRDA Level 2.

The remote LU must be able to obtain the verified end-user name associated with the conversation. Unless the verified end-user name is provided by DCE security mechanisms, DRDA requires the specification of SECURITY (PGM (USER_ID (*variable*) PASSWORD (*variable*))) or SECURITY(SAME) on ALLOCATE. The remote LU and the application server both use the authenticated USER_ID value for accounting purposes. The application server uses the authenticated USER_ID value to validate requester access to the remote database management system resources. Refer to Section 6.1 on page 270 and Section 6.1.1 on page 270 for further detail about architected end-user names.

SEND_DATA

Under normal circumstances, one or more SEND_DATA verbs follow ALLOCATE in a requester initialization verb sequence. The SEND_DATA verb transmits DDM commands and associated command data to the transaction program at the application server. The DDM commands that can flow identify the application requester and application server, establish requester and server capabilities, make relational database management system capabilities available to the requester, and request database management resources for processing a specific DRDA request.

Refer to Section 4.4.1 on page 54 for further detail on the DDM command sequences that DRDA uses.

The DATA parameter specifies the variable that contains the data to be sent.

RECEIVE Operations

Under normal circumstances after the last SEND_DATA, one or more RECEIVE_AND_WAIT or PREPARE_TO_RECEIVE, POST_ON_RECEIPT, TEST, and RECEIVE_AND_WAIT verb sequences must be performed.

An application requester initialization flow uses RECEIVE_AND_WAIT for a synchronous receive operation. The application requester uses a RECEIVE_AND_WAIT to receive DDM command reply objects including the execution results of application requester SQL statements.

An application requester initialization flow uses a sequence of PREPARE_TO_RECEIVE, POST_ON_RECEIPT, TEST, and RECEIVE_AND_WAIT verbs for an asynchronous receive operation. The use of POST_ON_RECEIPT and TEST allows the application requester to perform other types of processing before testing the conversation to determine whether reply object information is available for receipt. Checking for end-user keyboard interrupts is an example of one type of processing that the application requester can wish to perform. The application requester uses a RECEIVE_AND_WAIT to receive DDM command reply objects including the execution results of application requester SQL statements.

12.7.1.2 LU 6.2 Verbs that the Application Server Uses

The LU 6.2 verbs the application server uses for DRDA initialization flows are described here. Unless otherwise specified, refer to the *SNA Transaction Programmer's Reference Manual for LU Type 6.2* (GC30-3084, IBM) for further detail.

ATTACH Processing

LU 6.2 ATTACH processing in the communications product at the application server creates the resource ID. The manner in which a particular LU 6.2 communications product makes the resource ID available is specific to the environment.

GET_ATTRIBUTES

GET_ATTRIBUTES returns information about a conversation that the application server uses for request processing. This information includes the mode name, conversation state information,[54] and partner LU name that can be used for accounting purposes.

The RESOURCE parameter variable value for GET_ATTRIBUTES must specify the local resource ID of the conversation about which the application server desires information. The communications product at the application server creates the resource ID.

GET_TP_PROPERTIES

GET_TP_PROPERTIES returns information about the characteristics of the transaction program that the application server requires for request processing and that mechanisms specific to the environment can also use for accounting.

DRDA requires the SECURITY_USER_ID parameter. The SECURITY_USER_ID parameter specifies the variable for returning the architected end-user name carried on the allocation request that initiated the application requester initialization verb sequence. The application server requires the architected end-user name value for checking the requester's authorization to access database management system objects and for accounting purposes.

DRDA requires the LUW_IDENTIFIER or PROTECTED_LUW_IDENTIFIER[55] parameter. This parameter specifies the variable for returning the logical unit of work identifier associated with the transaction program. The application server can use the logical unit of work identifier for accounting mechanisms specific to the environment.

RECEIVE Operations

An application server initialization flow uses RECEIVE_AND_WAIT for each synchronous receive operation.

An application server initialization flow uses a sequence of POST_ON_RECEIPT, TEST, and RECEIVE_AND_WAIT verbs for each asynchronous receive operation. The use of POST_ON_RECEIPT and TEST allows the application server to perform other types of processing before testing the conversation to determine whether a DDM command or other information is available for receipt.

An application server uses a RECEIVE_AND_WAIT to receive a DDM command or the SEND indication. The application server can send data to the application requester only after it receives the SEND indication.

---

54. Conversation state information is useful for a transaction program to find out the state of the conversations prior to calling SYNCPT. This can help avoid state checks or help resolve a SYNCPT call that generated a state check.

55. For protected conversations in DRDA Level 2

SEND_DATA

Under normal circumstances, one or more SEND_DATA verbs follow a RECEIVE_AND_WAIT. The SEND_DATA verb transmits DDM command reply objects including the execution results of application requester SQL statements. The DATA parameter specifies the variable that contains the data to be sent.

12.7.1.3 Initialization Flows

The physical flow of information consists of a sequence of LU 6.2 verbs containing DDM commands.

Figure 12-1 on page 396 and Figure 12-3 on page 398 depicts DDM command processing using the LU 6.2 synchronous wait protocol verbs. DRDA also permits asynchronous wait protocols. Figure 12-1 on page 396 depicts the initialization flows while using LU 6.2 security. Figure 12-3 on page 398 depicts the initialization flows while using DCE security mechanisms. The primary difference between the two is the additional flows required to negotiate support for the security mechanism and then pass the DCE security context information which contains the end-user name and other security information.

An LU 6.2 ALLOCATE at the application requester causes the creation of a conversation between the application requester and application server. This conversation is allocated with SYNC_LEVEL(NONE) for DRDA Level 1 and can use SYNC_LEVEL(SYNCPT) for DRDA Level 2. Individual LU 6.2 SEND_DATA verbs at the application requester transmit each of the DDM request data stream structures for EXCSAT, ACCRDB, and EXCSQLSTT, along with any command data that the command can have. Individual LU 6.2 RECEIVE_AND_WAIT verbs at the application requester then receive the DDM reply data stream structure or object data stream structure response for each of the DDM commands. Other LU 6.2 RECEIVE_AND_WAIT verbs at the application requester receive the SEND indications.

An LU 6.2 GET_ATTRIBUTES and an LU 6.2 GET_TP_PROPERTIES at the application server obtain information about the conversation that is available to the application server following allocation. The obtained information includes the LUWID, mode, end-user name,[56] and partner LU name that the application server requires for request processing and accounting. Individual LU 6.2 RECEIVE_AND_WAIT verbs at the application server receive the DDM request data stream structures or command data. Other LU 6.2 RECEIVE_AND_WAIT verbs at the application server receive the SEND indications. Individual LU 6.2 SEND_DATA verbs at the server then transmit the DDM object data stream and reply data stream response structures for each of EXCSAT, ACCRDB, and EXCSQLSTT. LU 6.2 RECEIVE_AND_WAIT verbs at the application server cause the SEND indication to flow along with the contents of the SEND buffers.

Refer to Chapter 4 on page 37 for further detail about DRDA DDM command sequences.

The DRDA initialization flow while using LU 6.2 security consists of the following:

56. If DCE security mechanisms are in use, the end-user name provided in the DCE security context information take precedence over the end-user name provided in the LU 6.2 ALLOCATE flow.

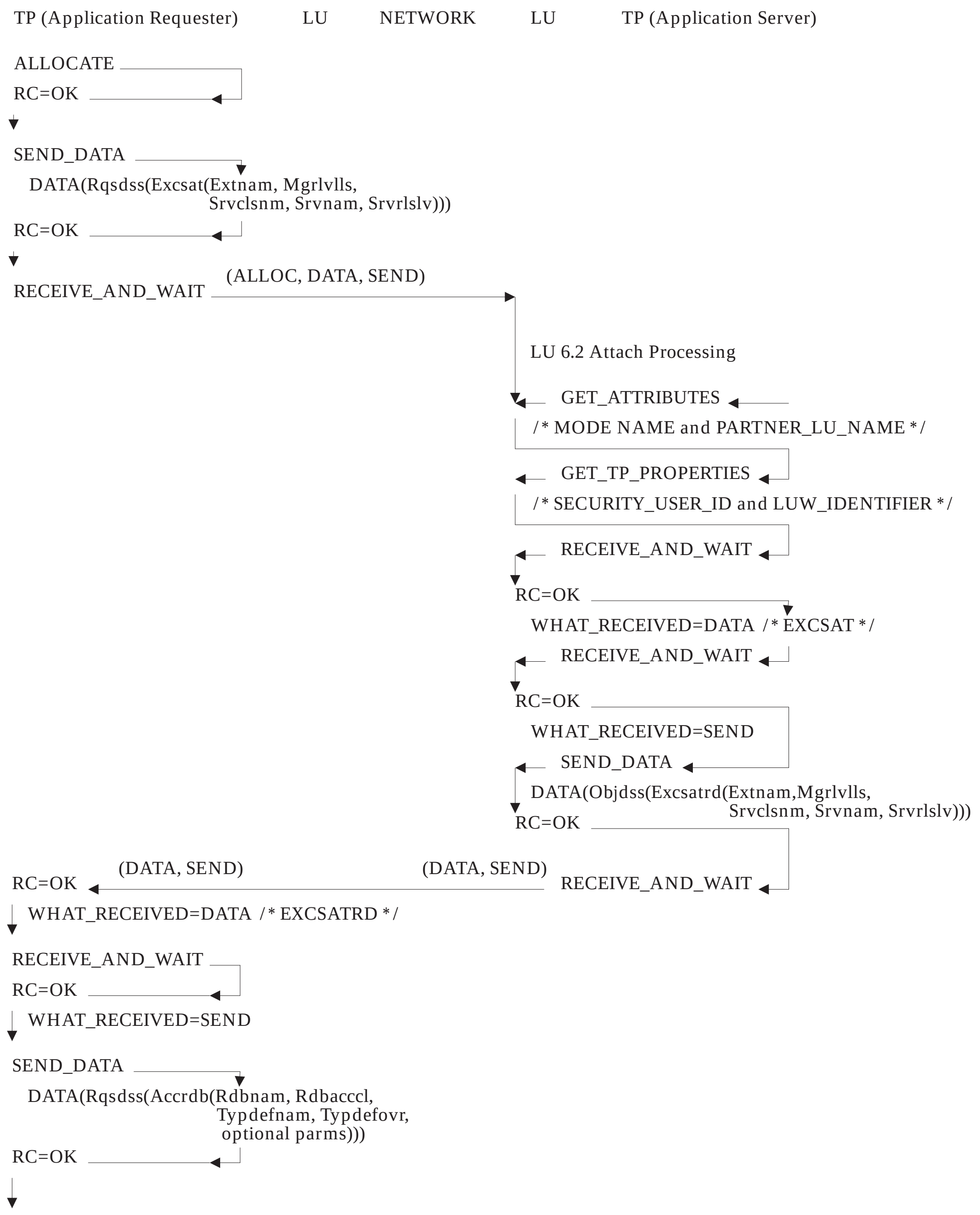

TP (Application Requester)
LU
NETWORK
LU
TP (Application Server)
ALLOCATE
RC=OK
SEND_DATA
DATA(Rqsdss(Excsat(Extnam, Mgrlvlls, Srvclsnm, Srvnam, Srvrlslv)))
RC=OK
RECEIVE_AND_WAIT
(ALLOC, DATA, SEND)
LU 6.2 Attach Processing
GET_ATTRIBUTES
/* MODE NAME and PARTNER_LU_NAME */
GET_TP_PROPERTIES
/* SECURITY_USER_ID and LUW_IDENTIFIER */
RECEIVE_AND_WAIT
RC=OK
WHAT_RECEIVED=DATA /* EXCSAT */
RECEIVE_AND_WAIT
RC=OK
WHAT_RECEIVED=SEND
SEND_DATA
DATA(Objdss(Excsatrd(Extnam,Mgrlvlls, Srvclsnm, Srvnam, Srvrlslv)))
RC=OK
(DATA, SEND)
(DATA, SEND)
RC=OK
RECEIVE_AND_WAIT
WHAT_RECEIVED=DATA /* EXCSATRD */
RECEIVE_AND_WAIT
RC=OK
WHAT_RECEIVED=SEND
SEND_DATA
DATA(Rqsdss(Accrdb(Rdbnam, Rdbacccl, Typdefnam, Typdefovr, optional parms)))
RC=OK

Figure 12-1 DRDA Initialization Flows with LU 6.2 Security (Part 1)

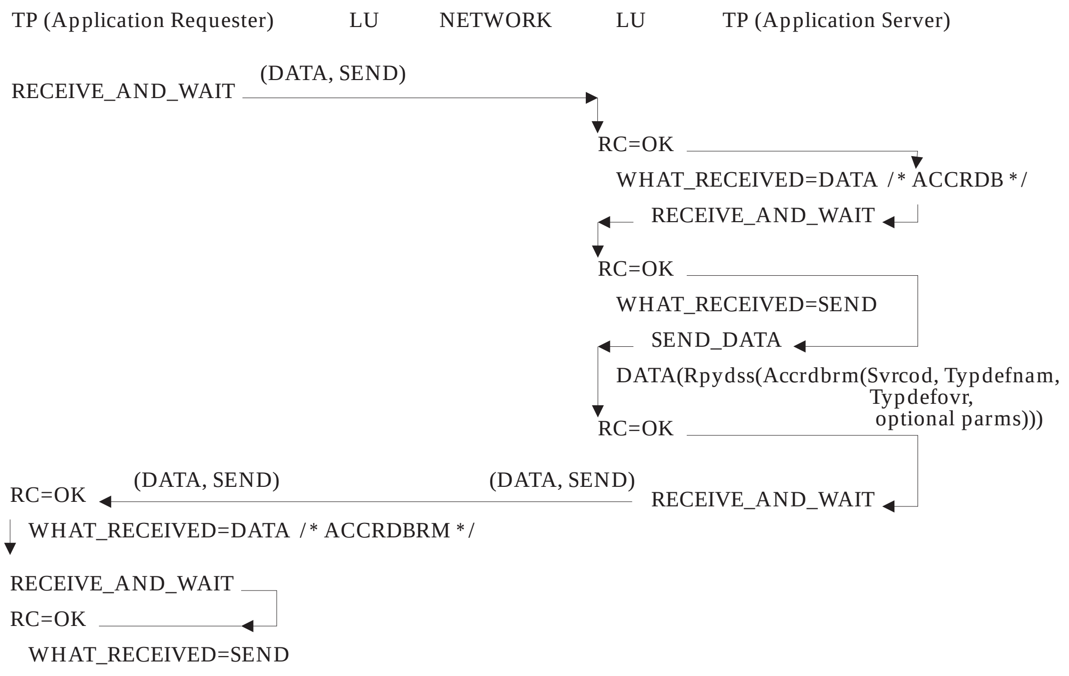


Figure 12-2 DRDA Initialization Flows with LU 6.2 Security (Part 2)

The DRDA initialization flow while using DCE security mechanisms is shown in Figure 12-3 on page 398.

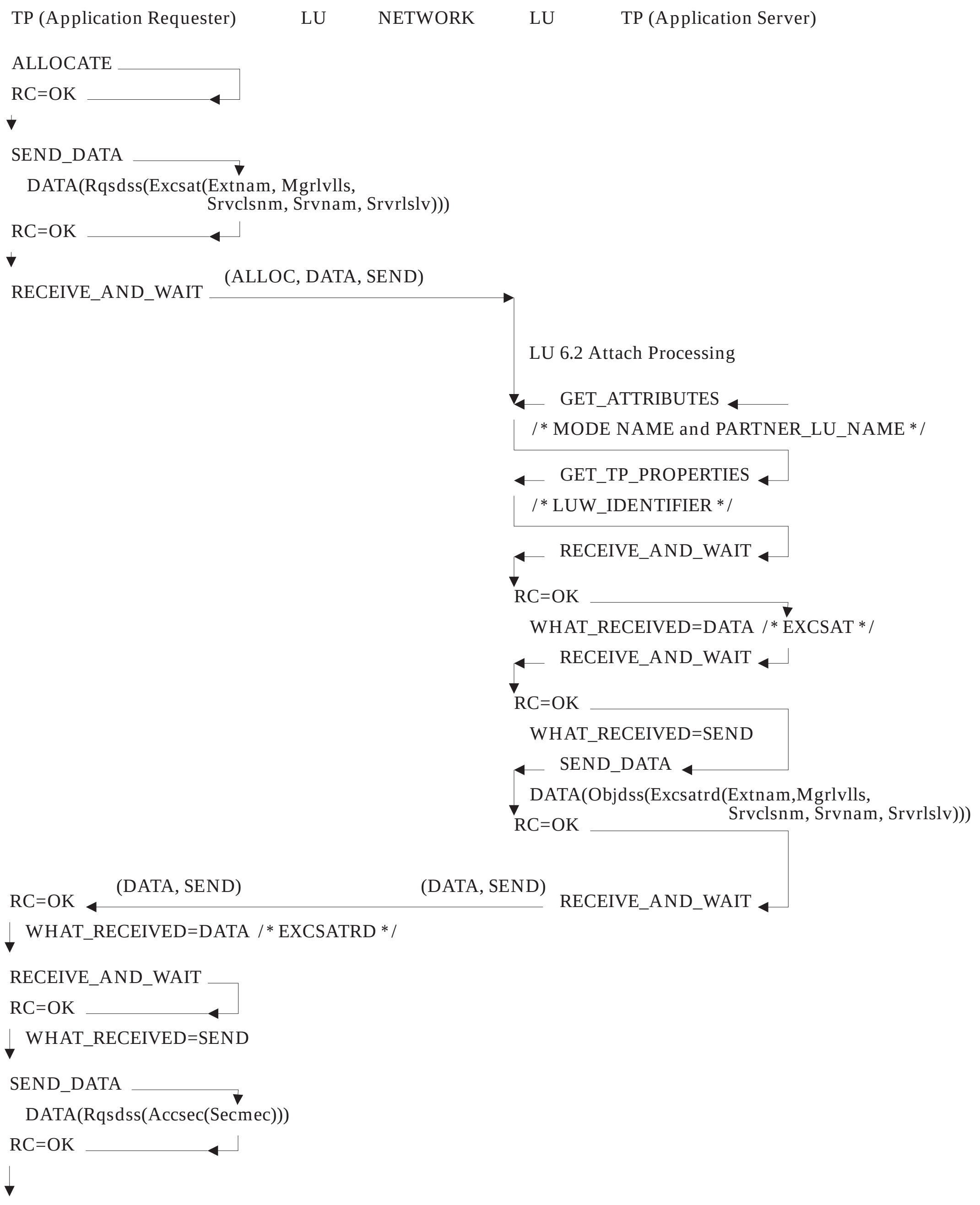


Figure 12-3 DRDA Initialization Flows with DCE Security (Part 1)

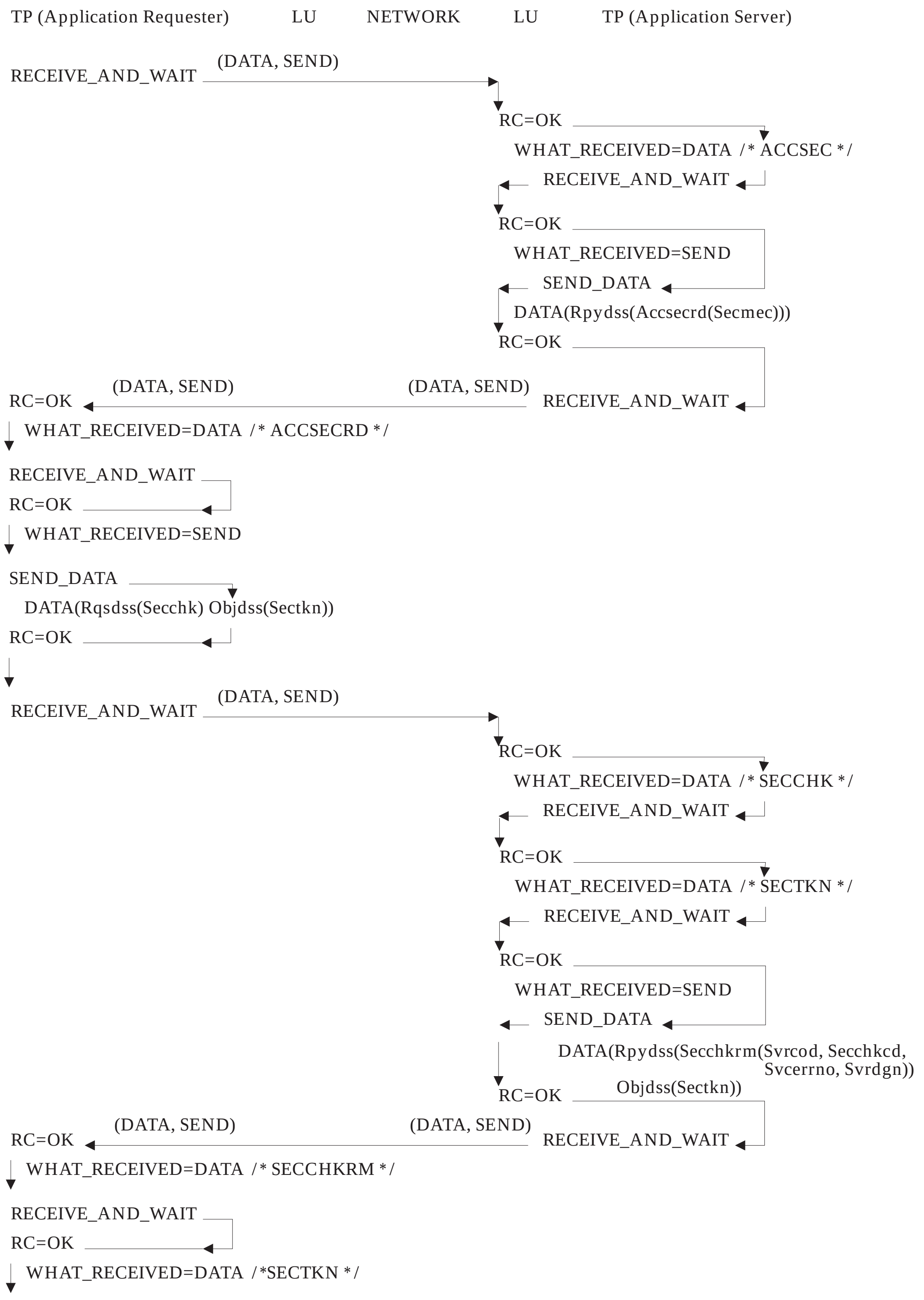


Figure 12-4 DRDA Initialization Flows with DCE Security (Part 2)

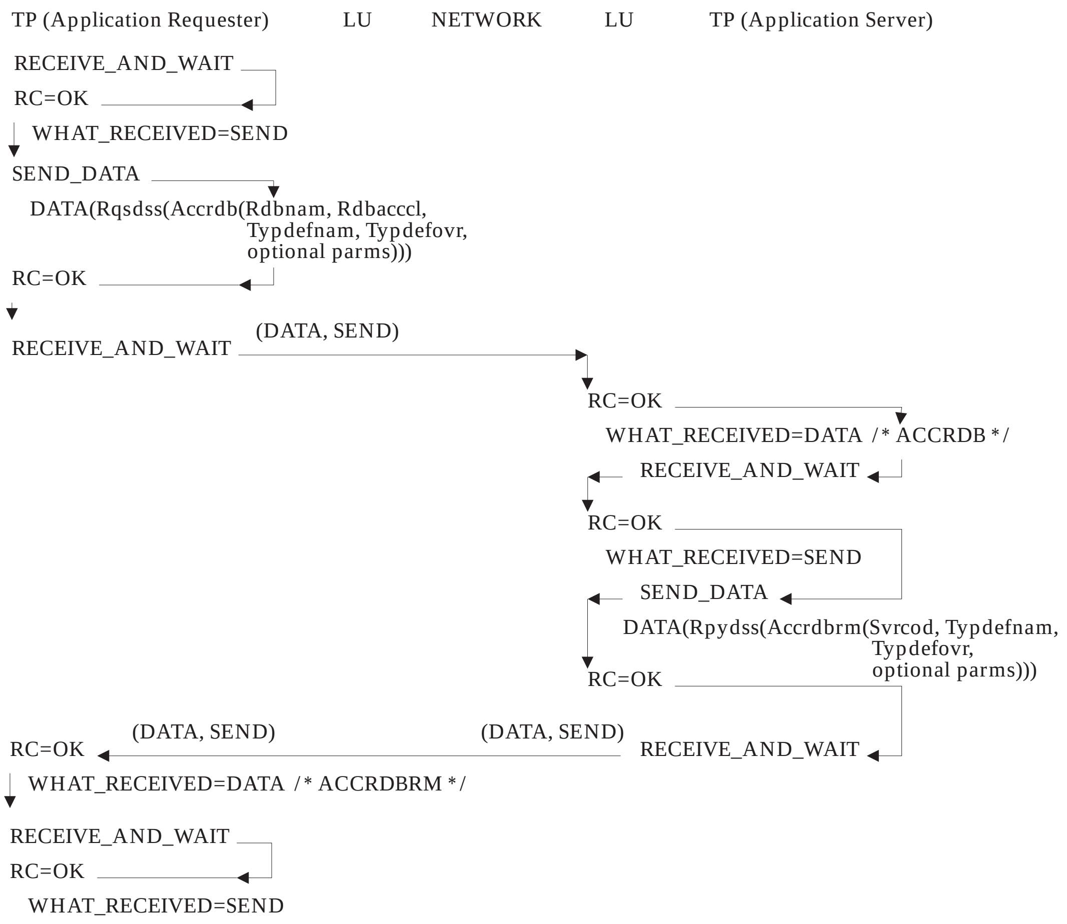


Figure 12-5 DRDA Initialization Flows with DCE Security (Part 3)

12.7.2 Processing a DRDA Request

DRDA requests exist for the processing of remote SQL statements and for the preparation of application programs. DRDA request flows transmit a remote DRDA request and its associated reply objects between an application requester and application server. Only an application requester can initiate a DRDA request flow.

Because authentication occurs during initialization processing, DRDA requires no additional authentication during DRDA request flows.

DRDA remote SQL statement requests often operate on multiple rows of multiple tables and can cause the transmission of multiple rows from the application server to the application requester. DRDA provides two data transfer protocols in support of these operations:

- Fixed-Row Protocol
- Limited Block-Protocol

Application requesters and application servers use the fixed-row protocol for the processing of a query that can be the target of a WHERE_CURRENT_OF clause on an SQL UPDATE or DELETE request, or for the processing of a multi-row fetch or fetch using a scrollable cursor. The fixed-row protocol guarantees the return of no more than the number of rows requested by the application whenever the application requester receives row data.

Application requesters and application servers use the limited block-protocol for the processing of a query that uses a cursor for read-only access to data. The limited block-protocol optimizes data transfer by guaranteeing the transfer of a minimum amount of data (which can be part of a row, multiple rows, or multiple rows and part of a row) in response to each DRDA request.

Refer to Section 4.4.6 on page 76 for further detail on DRDA data transfer protocols.

The DDM Reference provides a general overview of the component communications flows that comprise a DRDA request flow. The DDM terms APPSRCCR and APPSRCCD discuss synchronous requester and server communications flows that occur during the processing of a DRDA remote request.

12.7.2.1 LU 6.2 Verbs that the Application Requester Uses

The following discussion summarizes the LU 6.2 verbs the application requester uses for DRDA request flows.

Unless otherwise specified, see the *SNA Transaction Programmer's Reference Manual for LU Type 6.2* (GC30-3084, IBM) for further detail.

SEND_DATA

One or more SEND_DATA verbs initiate a requester DRDA request verb sequence. The SEND_DATA verb transmits DDM commands and command objects that request remote database management resources for processing a specific remote DRDA request.

The DATA parameter specifies the variable that contains the data to be sent. Refer to Section 4.4.3 on page 67through Section 4.4.11 on page 114 for further detail on the DDM command sequences that DRDA uses.

RECEIVE Operations

Under normal circumstances, either RECEIVE_AND_WAIT or a sequence of PREPARE_TO_RECEIVE, POST_ON_RECEIPT, TEST, and RECEIVE_AND_WAIT verbs must follow the SEND_DATA verb in an application requester DRDA request verb sequence.

12.7.2.2 LU 6.2 Verbs that the Application Server Uses

The following discussion summarizes the LU 6.2 verbs the application server uses for DRDA request flows.

Unless otherwise specified, see the *SNA Transaction Programmer's Reference Manual for LU Type 6.2* (GC30-3084, IBM) for further detail.

RECEIVE Operations

Under normal circumstances, either one or more RECEIVE_AND_WAIT verbs or one or more sequences of POST_ON_RECEIPT, TEST, and RECEIVE_AND_WAIT verbs initiate an application server DRDA request verb sequence.

SEND_DATA

Under normal circumstances, one or more SEND_DATA verbs follow the initial RECEIVE_AND_WAIT. The SEND_DATA verb transmits DDM command reply objects including the execution results of application requester SQL statements. The DATA parameter specifies the variable that contains the data to be sent.

12.7.2.3 Bind Flows

The physical flow of information consists of a sequence of LU 6.2 verbs containing DDM commands, FD:OCA data, SQL communication areas, and SQL statements.

Figure 12-6 on page 403 depicts DDM command processing using the LU 6.2 synchronous wait protocol verbs. DRDA also permits asynchronous wait protocols. Figure 12-6 on page 403 assumes that DDM command chaining is not being used.

Individual LU 6.2 SEND_DATA verbs at the application requester transmit each of the DDM request data stream structures for BGNBND, BNDSQLSTT, and ENDBND along with any command data that the command can have. Individual LU 6.2 RECEIVE_AND_WAIT verbs at the application requester then receive the DDM object data stream structure response for each of the DDM commands. Other LU 6.2 RECEIVE_AND_WAIT verbs at the application requester receive the SEND indications.

Individual LU 6.2 RECEIVE_AND_WAIT verbs at the application server receive each DDM request data stream structure or command data stream structure. Other LU 6.2 RECEIVE_AND_WAIT verbs at the application server receive the SEND indications. Individual LU 6.2 SEND_DATA verbs at the server then transmit the DDM object data stream and reply data stream response structures for each of BGNBND, BNDSQLSTT, and ENDBND. LU 6.2 RECEIVE_AND_WAIT verbs at the application server cause the SEND indication to flow along with the contents of the SEND buffers.

Refer to Chapter 4 on page 37 for further detail about DRDA DDM command sequences.

A bind flow is shown in Figure 12-6 on page 403.

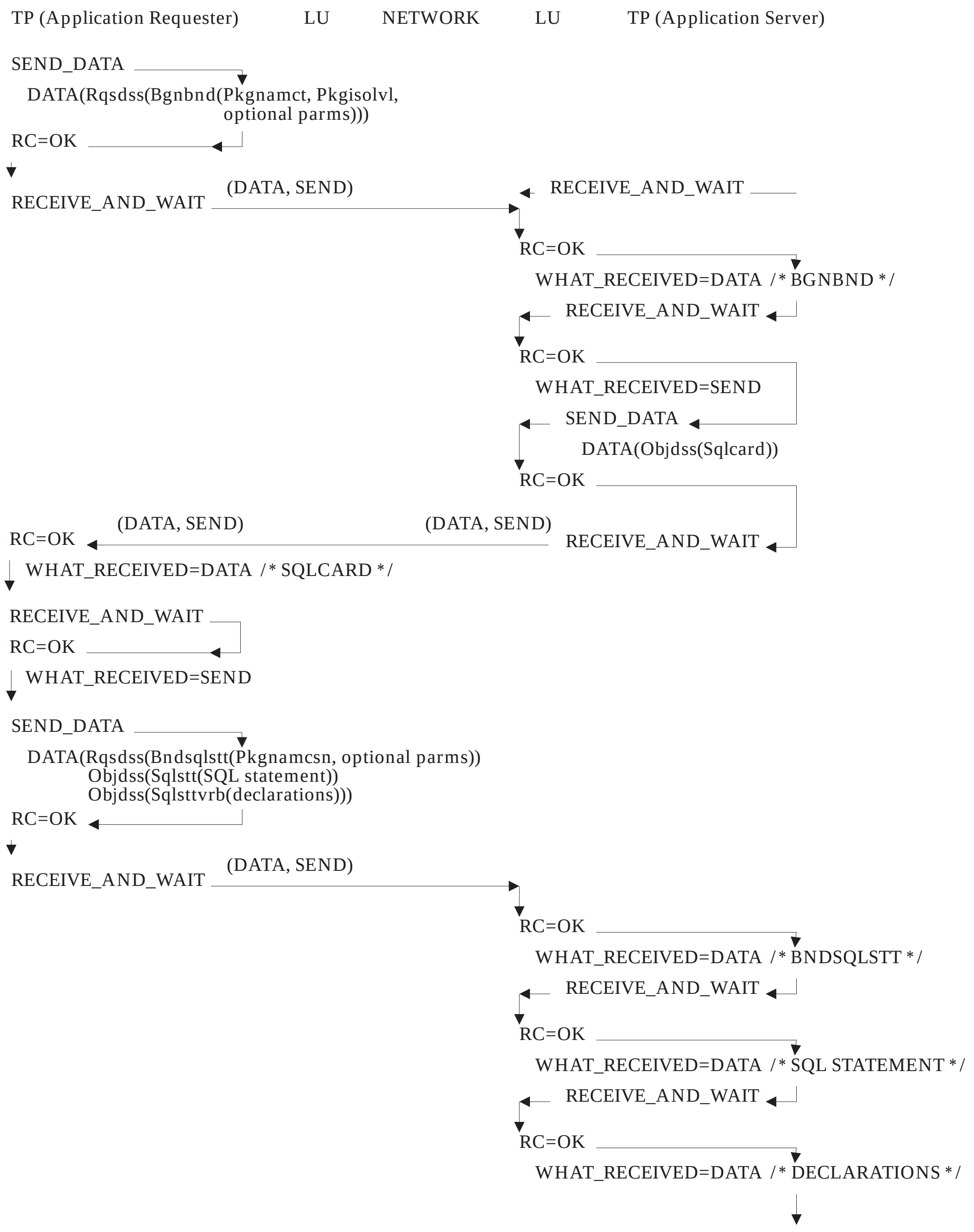

TP (Application Requester)
LU
NETWORK
LU
TP (Application Server)
SEND_DATA
DATA(Rqsdss(Bgnbnd(Pkgnamct, Pkgisolvl, optional parms)))
RC=OK
RECEIVE_AND_WAIT
(DATA, SEND)
RECEIVE_AND_WAIT
RC=OK
WHAT_RECEIVED=DATA /* BGNBND */
RECEIVE_AND_WAIT
RC=OK
WHAT_RECEIVED=SEND
SEND_DATA
DATA(Objdss(Sqlcard))
RC=OK
(DATA, SEND)
(DATA, SEND)
RC=OK
RECEIVE_AND_WAIT
WHAT_RECEIVED=DATA /* SQLCARD */
RECEIVE_AND_WAIT
RC=OK
WHAT_RECEIVED=SEND
SEND_DATA
DATA(Rqsdss(Bndsqlstt(Pkgnamcsn, optional parms))
Objdss(Sqlstt(SQL statement))
Objdss(Sqlsttvrb(declarations)))
RC=OK
RECEIVE_AND_WAIT
(DATA, SEND)
RC=OK
WHAT_RECEIVED=DATA /* BNDSQLSTT */
RECEIVE_AND_WAIT
RC=OK
WHAT_RECEIVED=DATA /* SQL STATEMENT */
RECEIVE_AND_WAIT
RC=OK
WHAT_RECEIVED=DATA /* DECLARATIONS */

Figure 12-6 DRDA Bind Flows (Part 1)

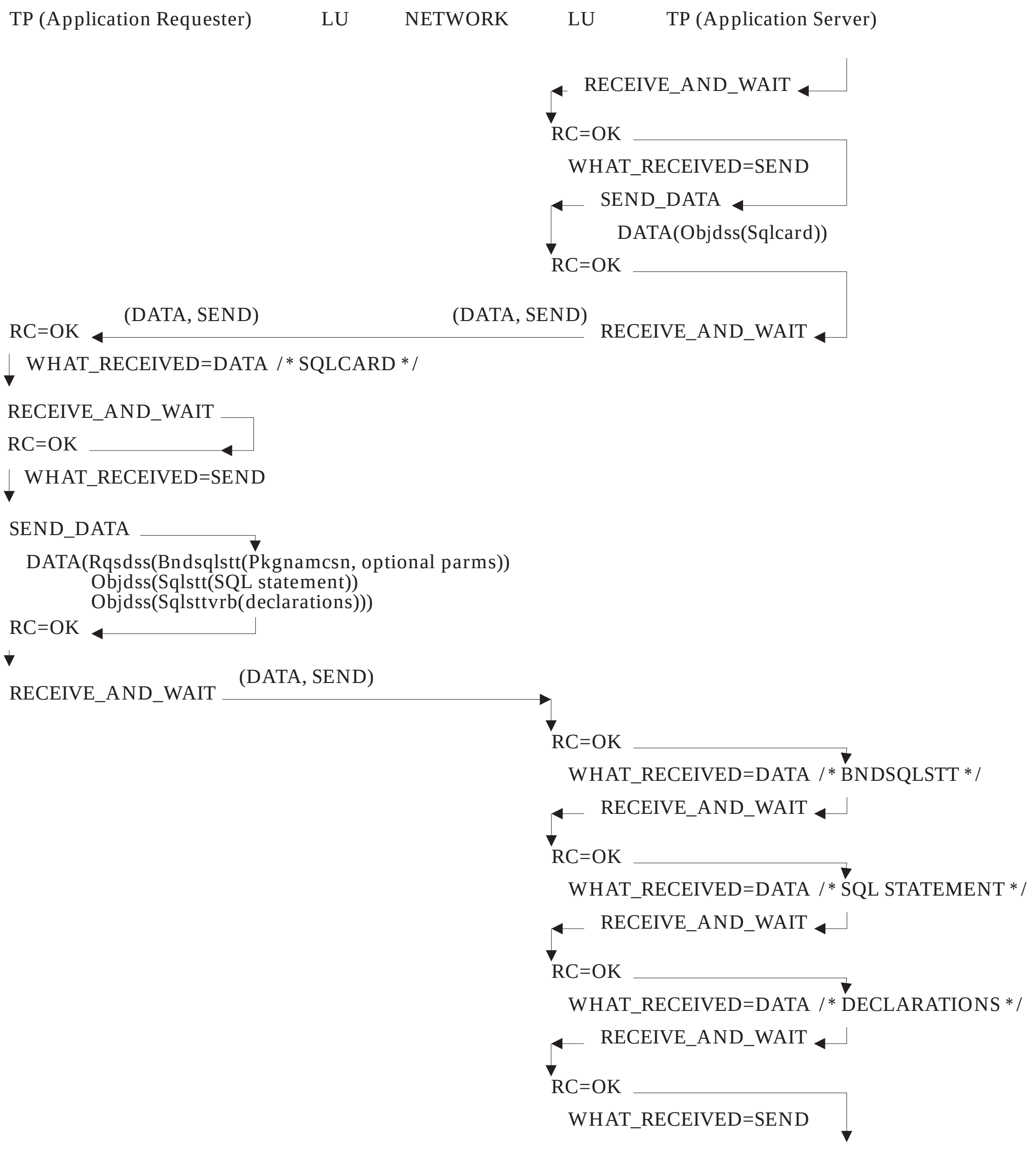


Figure 12-7 DRDA Bind Flows (Part 2)

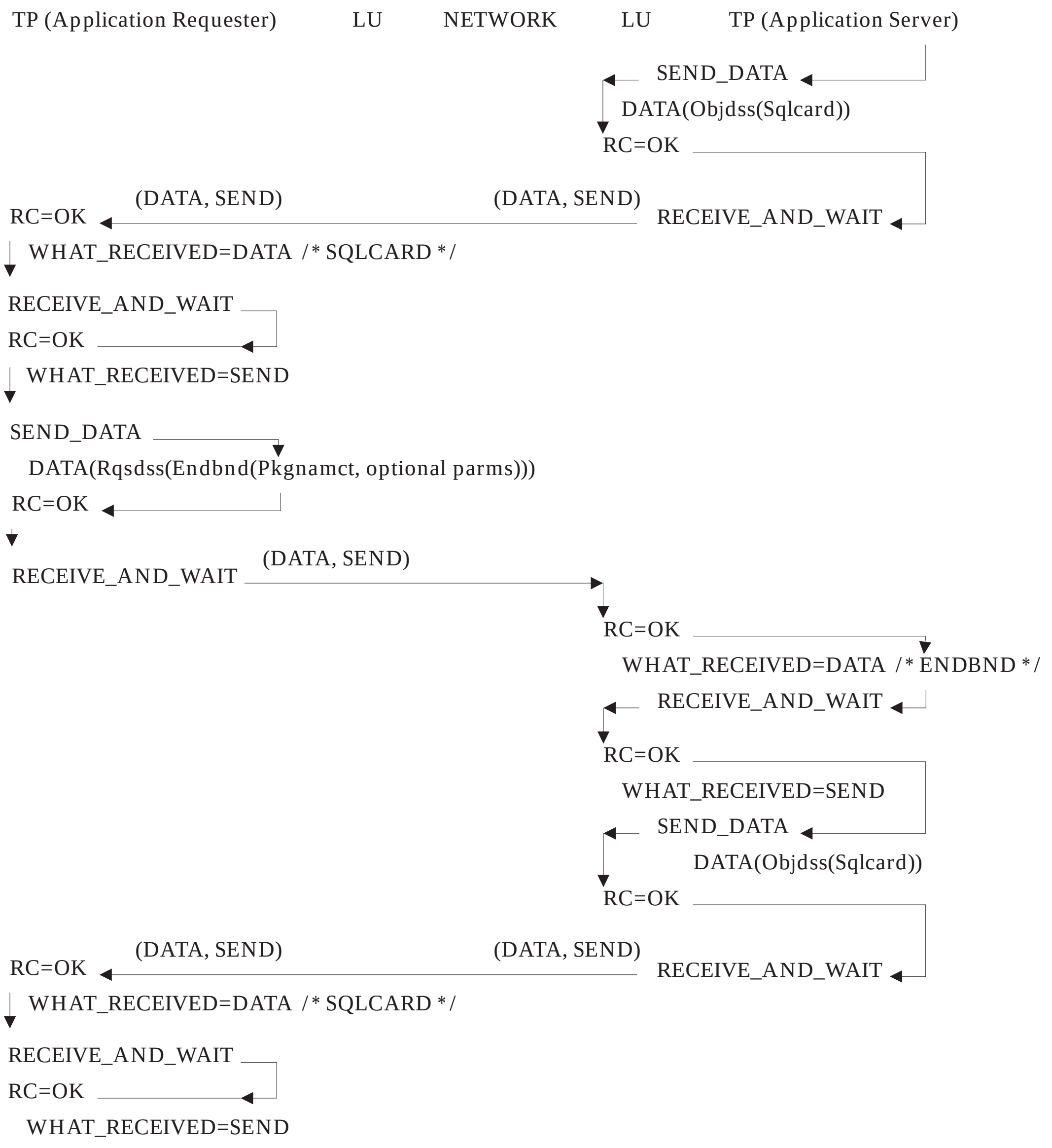


Figure 12-8 DRDA Bind Flows (Part 3)

12.7.2.4 *SQL Statement Execution Flows*

Figure 12-9 on page 406 depicts DDM command processing using the LU 6.2 synchronous wait protocol verbs. DRDA also permits asynchronous wait protocols.

The physical flow of information consists of a sequence of LU 6.2 verbs containing DDM commands, FD:OCA data descriptors, FD:OCA data, and DDM reply messages.

Individual LU 6.2 SEND_DATA verbs at the application requester transmit each of the DDM request data stream structures for OPNQRY and CNTQRY. Individual LU 6.2 RECEIVE_AND_WAIT verbs at the application requester then receive the DDM object data stream structure and reply message responses for the DDM commands. Other LU 6.2 RECEIVE_AND_WAIT verbs at the application requester receive the SEND indications.

Individual LU 6.2 RECEIVE_AND_WAIT verbs at the application server receive each DDM request data stream structure. Other LU 6.2 RECEIVE_AND_WAIT verbs at the application server receive the SEND indications. Individual LU 6.2 SEND_DATA verbs at the application server then transmit the DDM object data stream and reply message response structures for each of OPNQRY and CNTQRY. LU 6.2 RECEIVE_AND_WAIT verbs at the application server cause the SEND indication to flow along with the contents of the SEND buffers.

Refer to Chapter 4 on page 37 for further detail about DRDA DDM command sequences.

Figure 12-9 shows the SQL statement execution flow.

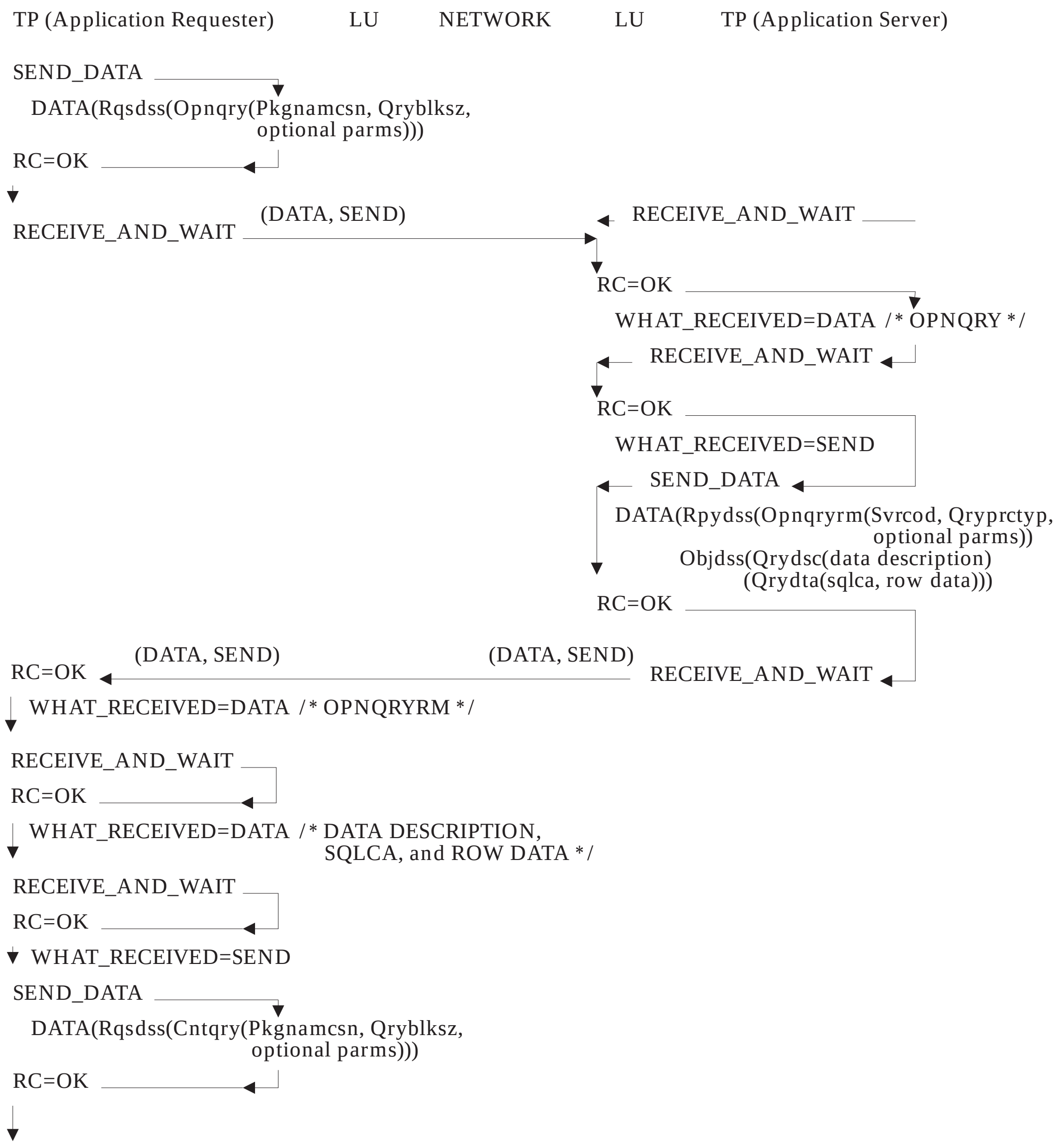

TP (Application Requester)
LU
NETWORK
LU
TP (Application Server)
SEND_DATA
DATA(Rqsdss(Opnqry(Pkgnamcsn, Qryblksz, optional parms)))
RC=OK
RECEIVE_AND_WAIT
(DATA, SEND)
RECEIVE_AND_WAIT
RC=OK
WHAT_RECEIVED=DATA /* OPNQRY */
RECEIVE_AND_WAIT
RC=OK
WHAT_RECEIVED=SEND
SEND_DATA
DATA(Rpydss(Opnqryrm(Svrcod, Qryprctyp, optional parms))
Objdss(Qrydsc(data description)
(Qrydta(sqlca, row data)))
RC=OK
(DATA, SEND)
(DATA, SEND)
RC=OK
RECEIVE_AND_WAIT
WHAT_RECEIVED=DATA /* OPNQRYRM */
RECEIVE_AND_WAIT
RC=OK
WHAT_RECEIVED=DATA /* DATA DESCRIPTION, SQLCA, and ROW DATA */
RECEIVE_AND_WAIT
RC=OK
WHAT_RECEIVED=SEND
SEND_DATA
DATA(Rqsdss(Cntqry(Pkgnamcsn, Qryblksz, optional parms)))
RC=OK

Figure 12-9 DRDA SQL Statement Execution Flows (Part 1)

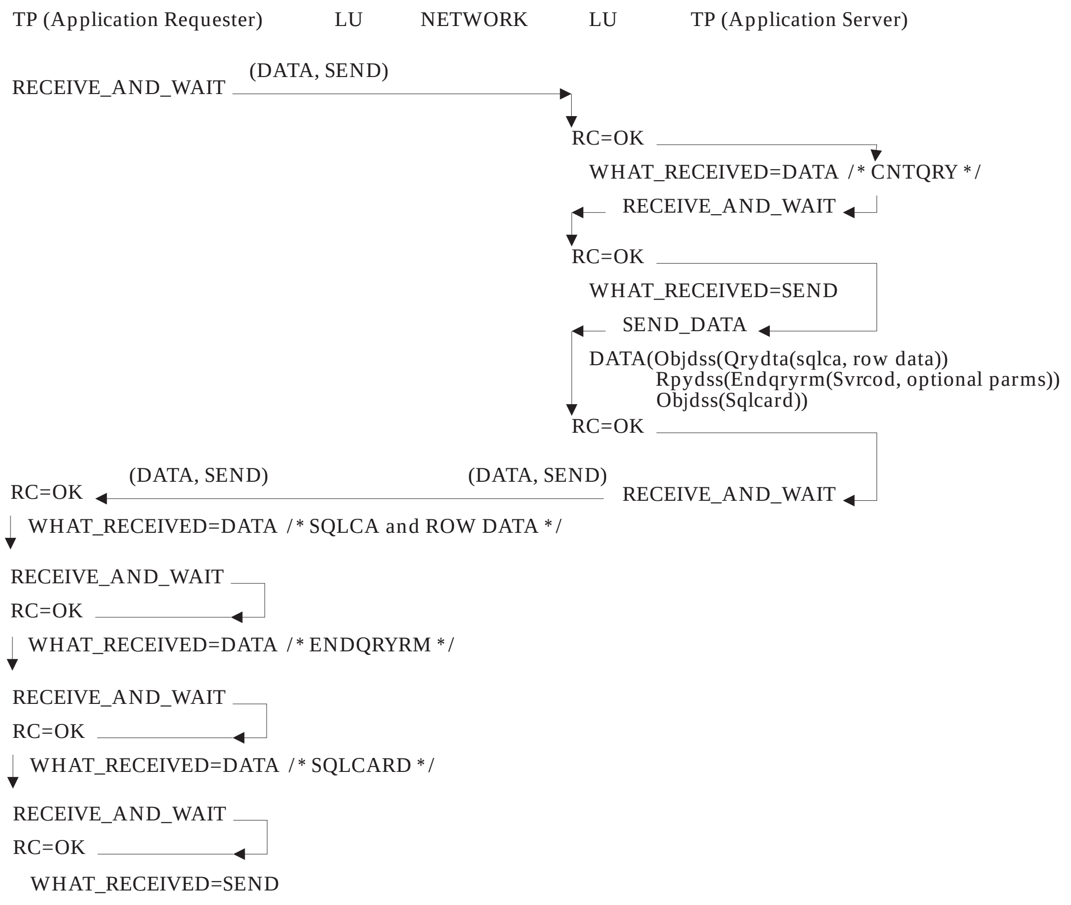


Figure 12-10 DRDA SQL Statement Execution Flows (Part 2)

12.7.3 Terminating a Conversation

Terminate conversation processing deallocates a conversation thereby making the conversation resources, including the underlying session, available for reuse at both the application requester and application server. Under normal circumstances, only an application requester terminates a conversation. In error situations, an application server can also terminate a conversation.

The deallocation of the conversation between an application requester and an instance of an application server terminates the communications between the application requester and that instance of the application server.

The application requester must ensure that all conversations associated with the execution of the application are terminated when the application normally or abnormally terminates.

On a SYNC_LEVEL(NONE) conversation, a DEALLOCATE flows to the application server. The DEALLOCATE includes an implied rollback at the application server. It is the responsibility of the application server to ensure a rollback during local deallocation processing at the application server.

On a SYNC_LEVEL(SYNCPT) conversation, the deallocation of the conversation is tied to resource recovery processing. The DEALLOCATE flows with the LU 6.2 two-phase commit protocols. If the logical unit of work rolls back, the conversation remains allocated. There is no implied rollback for application servers on SYNC_LEVEL(SYNCPT) conversations. An unconditional DEALLOCATE with a rollback must have a DEALLOCATION TYPE of ABEND_*.

An application requester might not be able to issue a DEALLOCATE with TYPE of SYNC_LEVEL prior to the beginning of resource recovery processing as a result of application termination. The application requester must terminate the conversations after the initial resource recovery process completes.

The DDM Reference provides a general overview of the communications flows that comprise a DRDA terminate conversation flow. The DDM terms APPCMNT and SYNCMNT describe termination of LU 6.2 communications associated with a conversation.

12.7.3.1 LU 6.2 Verbs that the Application Requester Uses

The LU 6.2 verbs the application requester uses for DRDA terminate conversation flows are described here. Unless otherwise specified, refer to the *SNA Transaction Programmer's Reference Manual for LU Type 6.2* (GC30-3084, IBM) for more detail.

DEALLOCATE

DEALLOCATE deallocates a conversation from the application requester, and eventually causes the deallocation of the conversation from the application server.

The TYPE parameter value must be FLUSH or SYNC_LEVEL for normal deallocation of a conversation. Either FLUSH or SYNC_LEVEL specifies the execution of the function of the FLUSH verb and the deallocation of the conversation normally.

The LOG_DATA parameter value can be YES or NO. DRDA has no requirement to place product-unique error information in the system error logs of the LUs supporting this conversation.

SYNCPT

For conversations allocated SYNC_LEVEL(SYNCPT), the DEALLOCATE does not flow until a SYNCPT verb is issued. Only one SYNCPT verb is needed to cause the DEALLOCATE to flow on all conversations that were issued the DEALLOCATE(SYNC_LEVEL). SYNCPT begins the two-phase commit process, and if the logical unit of work successfully commits, the conversation is deallocated. If the logical unit of work rolls back, the conversation remains allocated.

12.7.3.2 LU 6.2 Verbs that the Application Server Uses

The LU 6.2 verbs the application server uses for DRDA terminate conversation flows are described here. Unless otherwise specified, refer to the *SNA Transaction Programmer's Reference Manual for LU Type 6.2* (GC30-3084, IBM) for further detail.

DEALLOCATE

A DEALLOCATE deallocates the conversation locally from the application server. The TYPE parameter must be LOCAL. A RECEIVE_AND_WAIT notifies the application server that an incoming deallocate request has arrived.

The LOG_DATA parameter value can be YES or NO. DRDA has no requirement to place product-unique error information in the system error logs of the LUs supporting this conversation.

SYNCPT

For conversations allocated SYNC_LEVEL(SYNCPT), the SYNCPT verb is issued in response to WHAT_RECEIVED=TAKE_SYNCPT_DEALLOCATE from a RECEIVE_AND_WAIT call. If the logical unit of work commits successfully, the application server issues a DEALLOCATE(LOCAL). If the logical unit of work backs out, the conversation remains allocated.

12.7.3.3 Termination Flow—SYNC_LEVEL(NONE) Conversation

The physical flow of information consists of one LU 6.2 verb. An LU 6.2 DEALLOCATE at the application requester causes the deallocation of a conversation between the application requester and application server.

A RECEIVE_AND_WAIT at the application server receives the deallocate request, which causes local deallocation of the conversation. Figure 12-11 shows the termination flow on a SYNC_LEVEL(NONE) conversation.

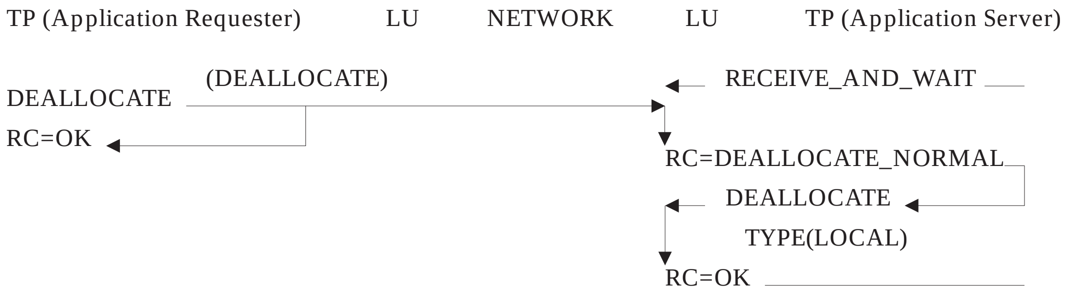


Figure 12-11 Actual Flow: Termination Flows on SYNC_LEVEL(NONE) Conversation

12.7.3.4 Termination Flow—SYNC_LEVEL(SYNCPT) Conversation

Figure 12-12 on page 410 displays the flows involved with deallocating a SYNC_LEVEL(SYNCPT) conversation. The flows are a simplified view of the two-phase commit synchronization point process. The LU and sync point manager (SPM) function are combined to avoid indicating the function split between the LU and the SPM. In practice, the LU and sync point manager share the responsibility to complete the two-phase commit protocol flows. For an in-depth description of the flows and the participating components, see the LU 6.2 documentation listed in **Referenced Documents** on page xxiv.

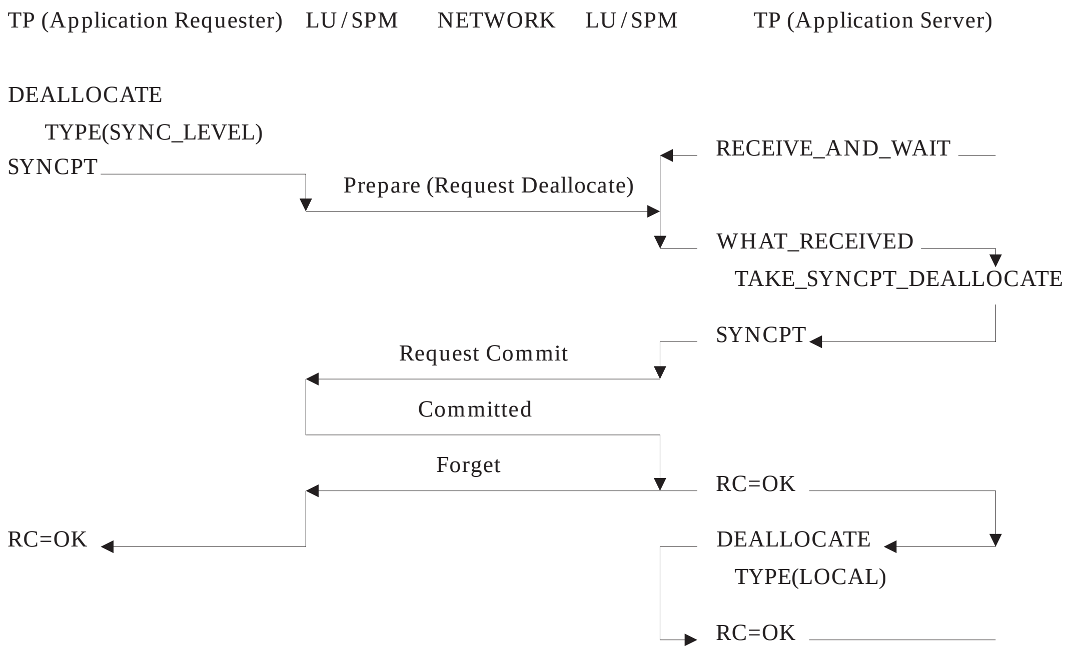


Figure 12-12 Actual Flow: Termination Flows on SYNC_LEVEL(SYNCPT) Conversation

12.7.4 Commit Flows on SYNC_LEVEL(NONE) Conversations

The physical flow of information for commit processing on SYNC_LEVEL(NONE) conversations consists of a sequence of LU 6.2 verbs containing DDM commands, and DDM reply messages.

An LU 6.2 SEND_DATA verb at the application requester transmits the DDM request data stream structure for RDBCMM. An LU 6.2 RECEIVE_AND_WAIT verb at the application requester then receives the DDM object data stream structure and reply message response for the DDM command. An LU 6.2 RECEIVE_AND_WAIT verb at the application requester receives the SEND indication.

An LU 6.2 RECEIVE_AND_WAIT verb at the application server receives the DDM request data stream structure. An LU 6.2 RECEIVE_AND_WAIT verb at the application server receives the SEND indication. An LU 6.2 SEND_DATA verb at the application server then transmits the DDM object data stream and reply message response structure for the RDBCMM. An LU 6.2 RECEIVE_AND_WAIT verb at the application server causes the SEND indication to flow with the contents of the SEND buffers.

Refer to Chapter 4 on page 37 for further detail about DRDA DDM command sequences.

Figure 12-13 on page 411 shows the commit execution flow.

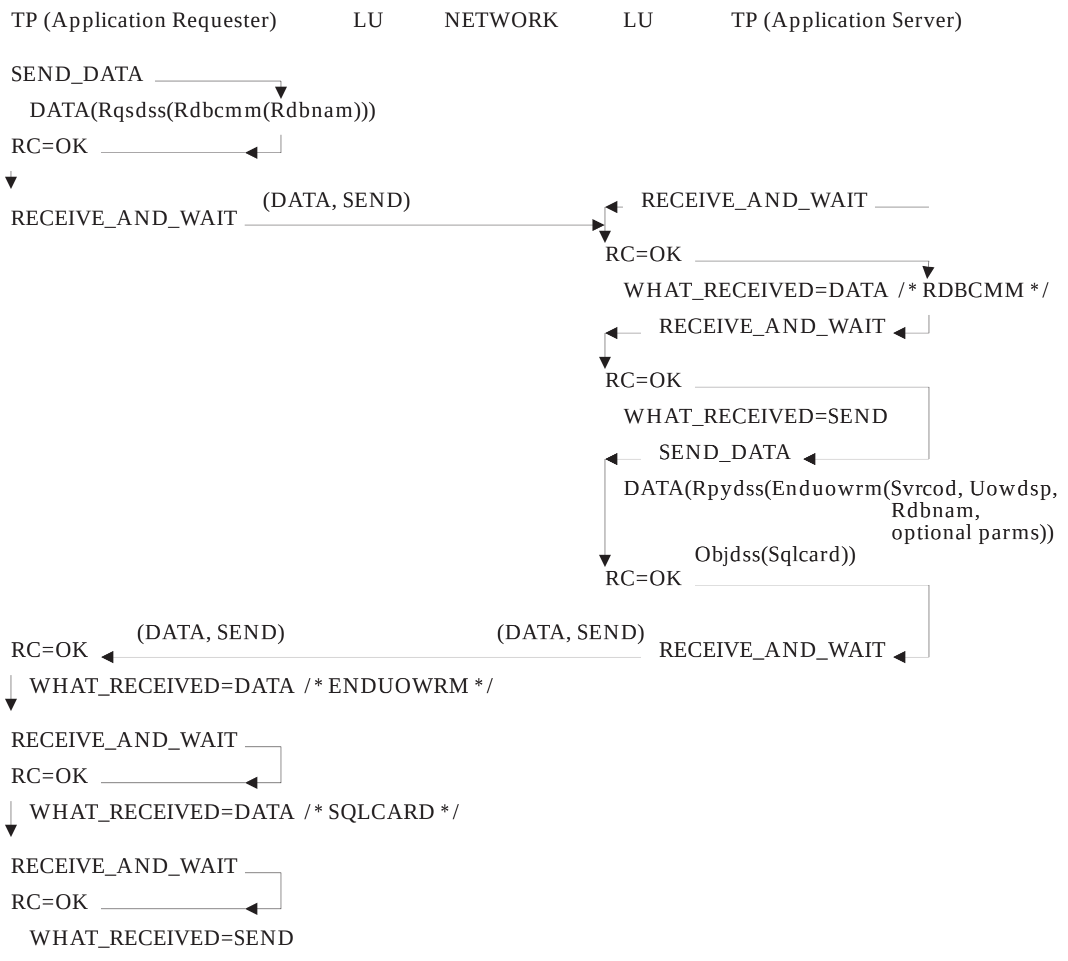


Figure 12-13 Commit Flow for a SYNC_LEVEL(NONE) Conversation

12.7.5 Rollback Flows on SYNC_LEVEL(NONE) Conversations

The physical flow of information for rollback processing on SYNC_LEVEL(NONE) conversations is the same as the commit flows on SYNC_LEVEL(NONE) conversations. See Section 12.7.4 on page 410 and replace RDBCMM with RDBRLLBCK.

12.7.6 Commit Flows on SYNC_LEVEL(SYNCPT) Conversations

Figure 12-14 on page 412 displays the flows involved with committing the logical unit of work on a SYNC_LEVEL(SYNCPT) conversation. The flows are a simplified view of the two-phase commit synchronization point process. The LU and sync point manager (SPM) functions are combined to avoid indicating the function split between the LU and the SPM. In practice, the LU and the sync point manager share the responsibility to complete the two-phase commit protocol flows. For an in-depth description of the flows and the participating components, see the LU 6.2 and documentation listed in **Referenced Documents** on page xxiv.

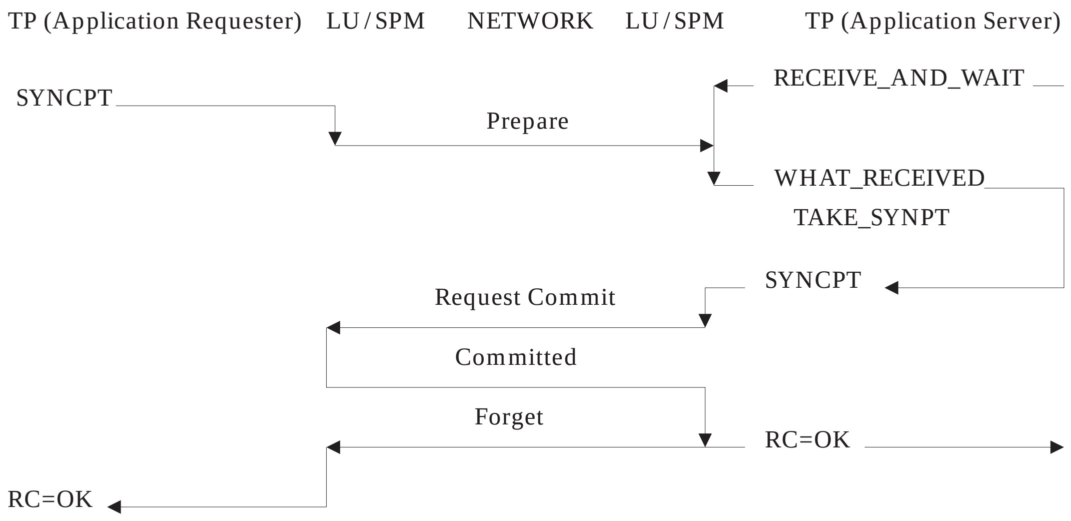


Figure 12-14 Actual Flow: Commit Flow on a SYNC_LEVEL(SYNCPT) Conversation

12.7.7 Rollback Flows on SYNC_LEVEL(SYNCPT) Conversations

Figure 12-15 displays the flows involved with rolling back the logical unit of work on a SYNC_LEVEL(SYNCPT) conversation. The flows are a simplified view of the synchronization point process. The LU and sync point manager (SPM) functions are combined to avoid indicating the function split between the LU and the SPM. In practice, the LU and the sync point manager share the responsibility to complete the synchronization point processing flows. For an in-depth description of the flows and the participating components see the LU 6.2 documentation listed in **Referenced Documents** on page xxiv.

If a relational database initiates a rollback, the flows described here would begin with a BACKOUT from the TP on the application server side.

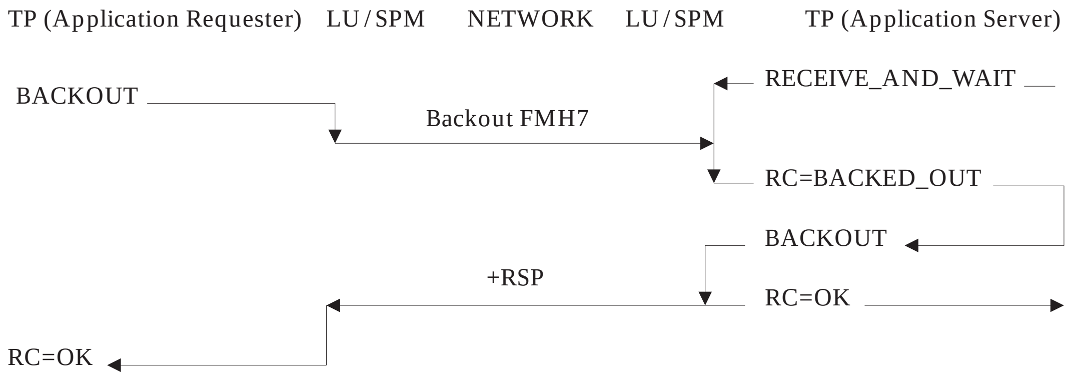


Figure 12-15 Actual Flow: Backout Flow on a SYNC_LEVEL(SYNCPT) Conversation

12.7.8 Handling Conversation Failures

LU 6.2 notifies both the application requester and the instance of the application server if the conversation linking the application requester to the instance of the application server fails. The application server must then implicitly roll back the effects of the application and deallocate all database management resources supporting the application. In the case of a failure on a SYNC_LEVEL(SYNCPT) conversation, the application requester and application server are placed in a backout required state by the local LU. The application requester and application server must issue a BACKOUT on the LU 6.2 interface. The application requester is also responsible for rolling back the application servers that are not on SYNC_LEVEL(SYNCPT) conversations.

In the case of a failure on a SYNC_LEVEL(NONE) conversation, the application requester is responsible for rolling back all other resources involved in the logical unit of work. If there are SYNC_LEVEL(SYNCPT) conversations, the application requester is responsible for issuing a BACKOUT on the LU 6.2 interface.

After all resources are rolled back, the application requester must report the failure to the application in the SQLCA. The application requester can then take one of two actions:

- Reject any subsequent SQL request from the application.
- Treat the next SQL request from the application as the beginning of a new unit of work. In this case, it would begin the DRDA initialization sequence again.

If there is a conversation failure in the middle of two-phase commit processing, the application server and application requester are waiting to regain control from the SYNCPT commands, so conversation failure at this time does not require special DRDA processing. The sync point manager initiates resync processing to complete the resource recovery process.

12.7.9 Managing Conversations Using Distributed Unit of Work

In a distributed unit of work environment, there can be several conversations involved in a logical unit of work. Proper management of the conversations provides performance benefits and optimizes the use of potentially limited conversation resources. The guideline for when a conversation can be deallocated is defined by the SQL semantics for SQL connections.

Due to coexistence and possible system restrictions, a SYNC_LEVEL(SYNCPT) conversation can be allocated to an application server that cannot operate at SYNC_LEVEL(SYNCPT). This can be prevented if the application server can identify to its local LU the SYNC_LEVEL the application server supports. If a SYNC_LEVEL(SYNCPT) conversation is successfully completed to an application server that does not support SYNC_LEVEL(SYNCPT), the application server will return a MGRDEPRM with *deperrcd* (01) at ACCRDB time. The application requester can allocate a new conversation with SYNC_LEVEL(NONE), and issue a DEALLOCATE(SYNC_LEVEL) on the SYNC_LEVEL(SYNCPT) conversation. The SYNC_LEVEL(SYNCPT) conversation will be deallocated at the next successful commit.

12.8 SNA Environment Usage in DRDA

This section describes considerations for problem determination in SNA environments, and rules usage and target program names usage in SNA environments.

12.8.1 Problem Determination in SNA Environments

The DRDA environment involves remote access to relational database management systems. Because the access is remote, enhancements to the local problem determination process were needed. These enhancements use Network Management tools and techniques. DRDA-required enhancements are alert generation with implied focal point support and a standard display for the logical unit of work identifier (LUWID). In DRDA Level 1, the LUWID is also used as a correlator between alerts and locally generated diagnostic information. In DRDA Level 2, the correlator between alerts and locally generated diagnostic information is the ACCRDB *crrtkn* parameter value.

12.8.1.1 LUWID

The logical unit of work identifier (LUWID) is defined to be unique, and is used as the correlator of information for DRDA Level 1. In DRDA Level 2, there can be two LUWIDs (protected and unprotected) involved, so the ACCRDB *crrtkn* parameter value is used as the correlator of information. If the application requester generates this value, it will use the value of the unprotected LUWID. See Section 11.2.2.2 on page 353 for more information on *crrtkn* and correlation.

12.8.1.2 DRDA LUWID and Correlation of Diagnostic Information

Because an LUWID plays an important role in correlation and work identification, DRDA specifies guidelines for LUWID display.

The LUWID is a network-wide unique identifier for a logical unit of work. The standardization of the display of the LUWID provides a consistent cross-product display. The LUWID display is in 2 forms. The short form is for informational displays that do not require recovery procedures. The long form includes a sequence number that helps in recovery procedures.

When displaying the short form of an LUWID, a product should include the fully qualified LUNAME and LUW instance number in the display.

The specific rules for the short form of an LUWID display are as follows:

1. Display the NETID.LUNAME portion of the LUWID as character data in NETID.LUNAME format (17 bytes maximum). The NETID and LUNAME are delimited by a period.
2. Display the LUW instance number as a string of hexadecimal characters (12 bytes total). The LUNAME and instance number are delimited by a period.

When displaying the long form of an LUWID, a product should include the fully qualified LUNAME, LUW instance number, and sequence number in the display.

The specific rules for the long form of an LUWID display are as follows:

1. Display the NETID.LUNAME portion of the LUWID as character data in NETID.LUNAME format (17 bytes maximum).
2. Display the LUW instance number as a string of hexadecimal characters (12 bytes total).
3. Display the sequence number as a string of hexadecimal characters (4 bytes total).

LUWIDs are Netid.Luname followed by instance number followed by the sequence number (if long form).

See ALLOCATE (Section 12.7.1.1 on page 392) for more information about LUWIDs.

12.8.1.3 Data Collection

When an error condition occurs at an application requester or application server, data should be gathered at that location. The data collection process should use the current tools available for the local environment. An application requester and application server must collect diagnostic information when they receive a reply message (RM) or generate an RM that falls into the category of the alerts defined in Table 11-1 on page 356. The application requester must gather diagnostic information when it receives an LU 6.2 DEALLOCATE with a type of ABEND on the conversation with the application server.

12.8.1.4 Alerts and Supporting Data in SNA Environments

Correlation between alerts and supporting data at each location, as well as cross-location, is done through correlation tokens. Using remote unit of work, the correlation token is the unprotected SNA LUWID. Using distributed unit of work, the correlation token is the ACCRDB *crrtkn* parameter value. The *crrtkn* value can be inherited at the application requester from the operating environment. If the inherited value matches the format of an SNA LUWID, then it is sent at ACCRDB in the *crrtkn* parameter. If the application requester does not inherit a correlation value, or the value does not match the format of an SNA LUWID, then the application requester must use the value of the unprotected LUWID, without the sequence number, as the *crrtkn* value. The correlation value is required in alerts and supporting diagnostic information.

The alert points to supporting data with subvector X'48' in the alert major vector. The data field in subfield X'85' for subvector X'48' must contain an identifier to the supporting data. This data field is an identifier of the supporting data. This identifier is location-dependent and must be good enough to uniquely identify the supporting data. Multiple subvector X'48's may be used in the alert. See the model in Table 11-3 on page 358 for more information on the subvector X'48'. See the *SNA Format and Protocol Reference Manual: Architecture Logic For LU Type 6.2* (SC30-3269, IBM) for a description of the alert subvectors.

12.8.2 Rules Usage for SNA Environments

This section consists of the SNA usage of the rules defined in Chapter 7 on page 281.

12.8.2.1 LU 6.2 Usage of Connection Allocation Rules

CA2 Usage

Conversations between an application requester and an application server must be basic conversations, TYPE(BASIC_CONVERSATION).

CA3 Usage

A conversation between an application requester and an application server using remote unit of work protocols must have SYNC_LEVEL(NONE).

A conversation between an application requester and an application server using distributed unit of work can have SYNC_LEVEL(NONE) or SYNC_LEVEL(SYNCPT). If either the application requester or application server does not support SYNC_LEVEL(SYNCPT), the conversation must have SYNC_LEVEL(NONE).

CA5 Usage

ACCRDB must be rejected with MGRDEPRM when DRDA-required LU 6.2 ALLOCATION parameters are not specified or are specified incorrectly.

The required LU 6.2 ALLOCATION parameters for ACCRDB in DRDA Level 1 are:

- TYPE(BASIC_CONVERSATION)
- SYNC_LEVEL(NONE)
- SECURITY(SAME) or
  SECURITY(PGM(USER_ID(variable))(PASSWORD(variable)))

  SECURITY(NONE) may be specified if user identification and authentication security is provided outside of the network.

The required LU 6.2 ALLOCATION parameters for ACCRDB using distributed unit of work protocols are:

- TYPE(BASIC_CONVERSATION)
- SYNC_LEVEL(NONE) or SYNC_LEVEL(SYNCPT)
- SECURITY(SAME) or
  SECURITY(PGM(USER_ID(variable)) (PASSWORD(variable)))

  SECURITY(NONE) may be specified if user identification and authentication security is provided using SECMGR Level 5. See rule SE2 usage in Section 7.11 on page 299.

12.8.2.2 LU 6.2 Usage of Commit/Rollback Processing Rules

CR2 Usage
Remote unit of work application servers and distributed unit of work application servers with SYNC_LEVEL(NONE) must inform the application requester when the current logical unit of work at the application server ends as a result of a commit or rollback request by an application or application requester request (dynamic commit and dynamic rollback are not allowed in distributed unit of work). This information is returned in the RPYDSS, containing the ENDUOWRM reply message. This RPYDSS is followed by an OBJDSS containing an SQLCARD with information that is input to the SQLCA to be returned to the application. If multiple commit or rollbacks occur prior to exiting a stored procedure, only one ENDUOWRM is returned. See rule CR13 in Section 7.4 on page 284 for setting the *uowdsp* parameter when multiple commit and/or rollbacks occur in a stored procedure. See CR6 for the SQLSTATEs to return.

CR8 Usage
An application server using distributed unit of work begins commit processing only if it is requested to commit. If an application requester receives an LU 6.2 TAKE_SYNCPT on the conversation with an application server, the application requester must ensure a rollback occurs for the logical unit of work.

DRDA Level 1 application requesters do not support the semantics of receiving TAKE_SYNCPT on the conversation.

12.8.2.3 LU 6.2 Usage of Security (SE Rules)

SE2 The application server must be able to obtain the verified end user name associated with the conversation. DRDA, therefore, requires one of the following mechanisms:

- The specification of one of the following LU 6.2-defined types of Conversation-Level Security on ALLOCATE:
  — SECURITY (PGM (USER_ID (variable) PASSWORD (variable) PROFILE (variable)))

The USER_ID value and the PASSWORD value must adhere to LU 6.2 access security information subfield constraints. The application server uses the PASSWORD value to verify the identity of the end user making the allocation request.

— SECURITY (SAME)

- The use of DCE-based security mechanisms for end-user identification and authentication.
- DRDA-defined security mechanisms for end-user identification and authentication.

ACCRDB must be rejected with MGRDEPRM if the application server does not obtain the verified end-user name.

SE3 If user identification and authentication security is not provided outside of the network, an application requester must have send support for each of the types of Conversation Level Security listed in rule SE2. An application server must have receive support for each of the types of Conversation-Level Security listed in rule SE2.

SE4 If user identification and authentication security is provided outside of the network, the security checks and values returned take precedence over the LU 6.2 security checks and values returned. For example, if an end-user name is provided on ALLOCATE, the end-user name supplied in the DCE security context information takes precedence over the end-user name received on ALLOCATE.

12.8.2.4 LU 6.2 Usage of Serviceability Rules

SV1 Usage

The application requester must generate diagnostic information and optionally generate an alert when it receives an LU 6.2 DEALLOCATE with a type ABEND from the application server.

SV8 Usage

The SNA LUWID or *crrtkn* or the ACCRDB must be present in the alert, in the supporting data information, and in diagnostic information.

SV9 Usage

Using distributed unit of work protocols, an application requester must send a correlation token to the application server at ACCRDB using the *crrtkn* parameter. If a correlation token exists for this logical unit of work, and it has the format of an SNA LUWID, then this token is used. If the existing token does not have the format of an SNA LUWID, or the token does not exist, then the application requester must send the SNA unprotected LUWID. The *crrtkn* value does not include the sequence number field of the LUWID.

12.8.2.5 LU 6.2 Usage of Names

This section describes usage of names for relational database names and for target program names.

LU 6.2 Usage of Relational Database Names Rules

RN2 Usage

DRDA associates an RDB_NAME with a specific transaction program name at a unique NETID.LUNAME. DRDA, however, does not define the mechanism that derives the NETID.LUNAME and transaction program name pair from the RDB_NAME. The particular derivation mechanisms are specific to the environment.

It is the responsibility of the application requester to determine the RDB_NAME name of the relational database and to map this name to an SNA logical unit name and transaction program name.

RN3 Usage

More than one RDB_NAME may exist for a single NETID.LUNAME. An RDB_NAME must map to a single NETID.LUNAME and Transaction Program Name.

RN4 Usage

DRDA permits the association of more than one RDB_NAME with a single transaction program name at a NETID.LUNAME.

LU 6.2 Usage of Transaction Program Names Rules

TPN1 Usage

The transaction program names identifying implemented DRDA application servers and database servers can be a registered DRDA transaction program name, a registered DDM transaction program name, or any non-registered transaction program name.

TPN2 Usage

DRDA allows DDM file servers and DRDA SQL servers to use either the same transaction program name or different transaction program names.

TPN3 Usage

Registered DRDA transaction program names begin with X‘07F6’. See the *SNA Transaction Programmer’s Reference Manual for LU Type 6.2* (GC30-3084, IBM) for details about registered transaction program names. DRDA transaction program names have a length of 4 bytes. The remaining characters of the transaction program name are Character Set 1134 A through Z and 0 through 9).

TPN4 Usage

Multiple DRDA transaction program names may exist for a single NETID.LUNAME

TPN5 Usage

A DRDA transaction program name is unique within an LU.

TPN6 Usage

Transaction programs (TPs) that are registered DRDA transaction program names must provide all the capabilities that DRDA requires.

TPN7 Usage

TPs that provide DRDA capabilities may perform additional non-DRDA TP work. These TPs are not required to perform additional non-DRDA TP work.

TPN8 Usage

The default DRDA transaction program name is X‘07F6C4C2’, and it is a registered transaction program name. The DRDA transaction program name X‘07F6C4C2’ must be definable at each LU that supports at least one application server providing DRDA capabilities.

12.8.3 Transaction Program Names

SNA LU 6.2 requires that an application requester (AR) specify the transaction program name of the application server (AS) when allocating a conversation. The application requester determines the transaction program name of the application server during the process of resolving the RDB_NAME of the application server to a NETID.LUNAME. DRDA allows the use of any valid transaction program name that meets the standards of the SNA transaction program name architecture and that the application server supports. Refer to the *SNA Format and Protocol Reference Manual: Architecture Logic For LU Type 6.2* (SC30-3269, IBM) and *SNA Transaction Programmer's Reference Manual for LU Type 6.2* (GC30-3084, IBM) for more details on transaction program name structure and use.

To avoid potential name conflicts, the application server transaction program name should be, but need not be, a registered SNA transaction program name. DRDA has defined one registered transaction program name that can be used. This transaction program name is X'07F6C4C2'. The first two bytes of this name (X'07F6') have been registered with SNA to represent the DRDA functional class for transaction programs. DRDA transaction programs are classified as SNA Service Transaction Programs because they provide SQL as the application interface rather than LU 6.2 verbs.

DDM also provides a registered transaction program name that can be used. This transaction program name is X'07F0F0F1'. The DDM transaction program name would be used if the DDM implementation at the application server provided file server functions in addition to DRDA functions.

The default DRDA transaction program name is X'07F6C4C2'. The DRDA transaction program name X'07F6C4C2' must be definable at each LU that has an application server providing DRDA capabilities. An application requester can then assume the existence of transaction program name X'07F6C4C2' at any LU providing DRDA capabilities, and default to transaction program name X'07F6C4C2' when a request requiring an ALLOCATE does not specify a transaction program name. Because transaction programs can have aliases, the transaction program with transaction program name X'07F6C4C2' can also have the DDM transaction program name X'07F0F0F1' or some other registered DRDA transaction program name. DRDA, however, does not require that a DRDA TP have multiple transaction program names.

Chapter 13

TCP/IP

This chapter summarizes the characteristics of DRDA communications flows using the TCP/IP network environment.

13.1 TCP/IP and the DDM Communications Model

Implementations of DRDA use the DDM Communications Managers. The TCP/IP Communications Manager (CMNTCPIP) supports the protocols defined by Transport Control Protocol/Internet Protocol (TCP/IP). For further detail, see the DDM terms CMNTCPIP in the DDM Reference.

13.2 What You Need to Know About TCP/IP

This chapter assumes some familiarity with TCP/IP and the sockets interface. The sockets interface is used only as a convenience to model the functionality level and calls to drive the TCP/IP protocols. With a general exposure to these topics, it should be possible to understand DRDA's use of TCP/IP. With more detailed knowledge, it should be possible to understand how to use TCP/IP in DRDA environments. For a list of relevant TCP/IP publications, see **Referenced Documents** on page xxiv.

The reader should also have some familiarity with DDM terms and the DDM model. A reader with a general exposure to DDM should be able to understand how DRDA's use of TCP/IP relates to the DDM communications managers of the DDM model.

Refer to **Referenced Documents** on page xxiv for the list of DDM publications.

13.3 TCP/IP

TCP/IP is made up of several parts that interact to provide network services to users. The parts are Applications Services, TCP, UDP, IP, and Network. These parts and their relationship to each other are graphically displayed in Figure 13-1. A brief description of the parts, follows the figure.

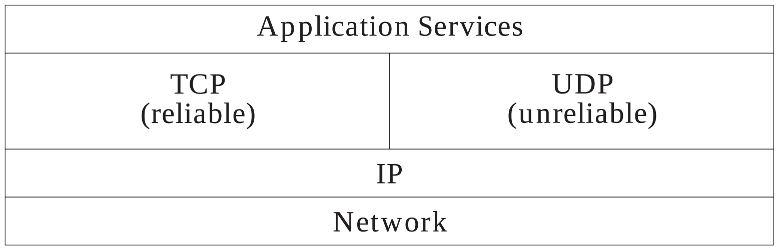


Figure 13-1 TCP/IP Components

13.3.1 Transport Control Protocol (TCP)

The transport control protocol is the level of service that DRDA needs to provide the integrity required by DRDA. TCP services on top of IP provide the required functions.

The interface between the application program and TCP can be characterized as:

- Stream-oriented

  The data is transferred between application programs in streams of bytes. The receiver receives the bytes in the same sequence as sent.

- Virtual Circuit Connection

  This is equivalent to a conversation in LU 6.2 terms. The applications are connected for the duration of the work and both sides of the TCP/IP connection are aware of the network address of the partner.

- Buffered Transfer

  The data can be buffered into packets independent of the pieces the application program transfers. The order of bytes is preserved and delivered in the same order sent.

- Unstructured Stream

  The structure of the data is known only by the applications involved in the TCP/IP connection. The applications must understand the stream content.

- Full Duplex Connection

  TCP/IP connections allow concurrent transfer in both directions. The SQL interface is synchronous, but DRDA can still take advantage of the full duplex feature. For example, an application server might begin returning answer set data before the application requester has completed sending a chain of commands, or an application requester may begin sending new commands before the application server has completed sending the answer set back from the previous command.

The reliability of TCP is provided by acknowledgments to the sender of a packet that the packet was received at the destination. The sent packet and acknowledgment contain a sequence number to test for duplication.

13.3.2 Application Services

The application services part is made up of high-level and specific services for applications. The application requester and application server are application services.

13.4 Sockets Interface

The sockets interface calls are defined in DRDA as a modeling tool to help describe the series of flows to drive DRDA protocol on a TCP/IP connection. Another interface might be chosen, but care should be taken to not introduce functions that are not supported at both ends of the TCP/IP connection.

13.5 TCP/IP and DRDA

Application requesters and application servers that provide DRDA capabilities use DRDA flows. DRDA flows permit implementations of DRDA to initialize TCP/IP connections, terminate TCP/IP connections, and process DRDA requests.

The socket calls that are of interest to DRDA are:

Socket	Creates an end point (socket) for communication.
Close	Closes a socket.
Bind	Establishes a local address for a socket.
Connect	Initiates a TCP/IP connection on a socket.
Listen	Listens for TCP/IP connection requests on a socket.
Accept	Accepts a TCP/IP connection on a socket.
Write	Sends data on a TCP/IP connection.
Read	Receives data on a TCP/IP connection.
Getpeername	Gets the address of the peer to which the socket connects.

13.5.1 Initializing a Connection

Initialization processing allocates a TCP/IP connection and prepares a DRDA execution environment. Only an application requester can start a TCP/IP connection. Authentication occurs during initialization processing through the use of DRDA flows. Database management systems verify that authenticated IDs have the authorization to perform DRDA database manager requests.

Refer to Section 6.1 on page 270 and Section 6.1.1 on page 270for a detailed description of architected end-user names.

Authentication between an application requester and application server occurs once per TCP/IP connection during DDM security manager Level 5 access security (ACCSEC) and security check (SECCHK) processing.

Initialization processing also propagates basic accounting information. The socket allows for the identification of the peer socket on the TCP/IP connection. The end-user name is derived from the SECCHK command. The correlation token is required to be passed when accessing the RDB as the *crrtkn* on the ACCRDB.

The DDM Reference provides a general overview of the component communications flows that comprise a DRDA initialization flow. See the DDM term TCPCMNI, which discusses initiation of TCP/IP connections.

13.5.1.1 Initialization Flows

The physical flow of information consists of a sequence of socket calls containing DDM commands. Figure 13-2 on page 425 depicts the initialization flows while using DRDA-defined userid and password security or DCE security mechanisms.

A socket call followed by a connect call at the application requester causes the creation of a TCP/IP connection between the application requester and application server. Individual write calls at the application requester transmit each of the DDM request data stream structures for EXCSAT, ACCRDB, and EXCSQLSTT, along with any command data that the command can have. Individual read calls at the application requester then receive the DDM reply data stream structure or object data stream structure response for each of the DDM commands.

Socket implementation-specific calls at the application server obtain information about the TCP/IP connection that is available to the application server. The obtained information includes the peer socket address. Individual read calls at the application server receive the DDM request data stream structures or command data. Individual write calls at the server then transmit the DDM object data stream and reply data stream response structures for each of EXCSAT, ACCRDB, and EXCSQLSTT.

Refer to Chapter 4 on page 37 for further detail about DRDA DDM command sequences.

The DRDA TCP/IP initialization flow with negotiation for security mechanisms consists of the following:

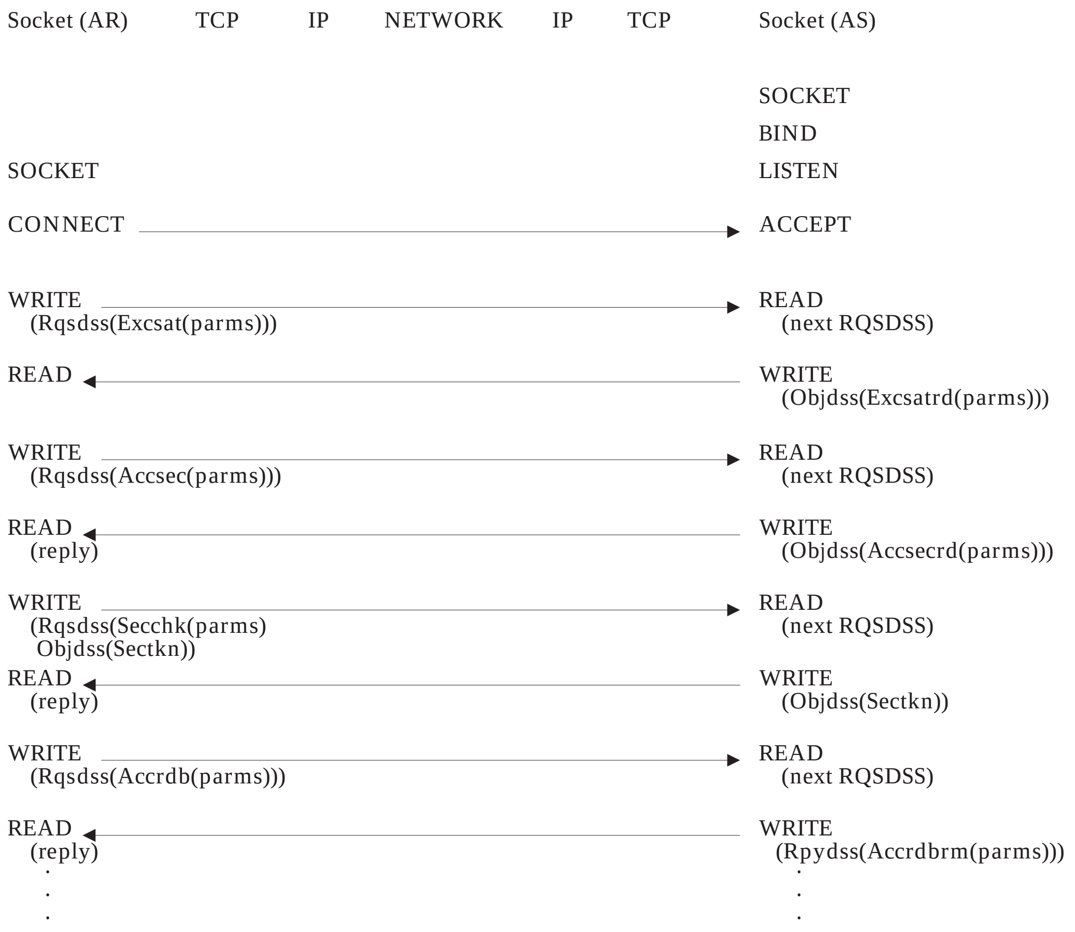

Socket (AR)
TCP
IP
NETWORK
IP
TCP
Socket (AS)
SOCKET
BIND
SOCKET
LISTEN
CONNECT
ACCEPT
WRITE (Rqsdss(Excsat(parms)))
READ (next RQSDSS)
READ
WRITE (Objdss(Excsatrd(parms)))
WRITE (Rqsdss(Accsec(parms)))
READ (next RQSDSS)
READ (reply)
WRITE (Objdss(Accsecrd(parms)))
WRITE (Rqsdss(Secchk(parms) Objdss(Sectkn))
READ (next RQSDSS)
READ (reply)
WRITE (Objdss(Sectkn))
WRITE (Rqsdss(Accrdb(parms)))
READ (next RQSDSS)
READ (reply)
WRITE (Rpydss(Accrdbrm(parms)))

Figure 13-2 DRDA Initialization Flows on TCP/IP with DCE Security

13.5.2 Processing a DRDA Request

DRDA requests exist for the processing of remote SQL statements and for the preparation of application programs. DRDA request flows transmit a remote DRDA request and its associated reply objects between an application requester and application server. Only an application requester can initiate a DRDA request flow.

Because authentication occurs during initialization processing, DRDA requires no additional authentication during DRDA request flows.

DRDA remote SQL statement requests often operate on multiple rows of multiple tables and can cause the transmission of multiple rows from the application server to the application requester. DRDA provides two data transfer protocols in support of these operations:

- Fixed-Row Protocol
- Limited Block-Protocol

Application requesters and application servers use the fixed-row protocol for the processing of a query that can be the target of a WHERE_CURRENT_OF clause on an SQL UPDATE or DELETE request, or for the processing of a multi-row fetch or fetch using a scrollable cursor. The fixed-row protocol guarantees the return of no more than the number of rows requested by the application whenever the application requester receives row data.

Application requesters and application servers use the limited block-protocol for the processing of a query that uses a cursor for read-only access to data. The limited block-protocol optimizes data transfer by guaranteeing the transfer of a minimum amount of data (which can be part of a row, multiple rows, or multiple rows and part of a row) in response to each DRDA request.

Refer to Section 4.4.6 on page 76 for further detail on DRDA data transfer protocols.

The DDM Reference provides a general overview of the component communications flows that comprise a DRDA request flow. The DDM terms TCPSRCCR and TCPSRCCD discuss requester and server communications flows that occur during the processing of a DRDA remote request.

13.5.2.1 Bind Flows

The physical flow of information consists of a sequence of packets containing DDM commands, FD:OCA data, SQL communication areas, and SQL statements.

Figure 13-3 on page 427 depicts DDM command processing using socket interface calls. Figure 13-3 on page 427 assumes that DDM command chaining is not being used.

Individual WRITE calls at the application requester transmit each of the DDM request data stream structures for BGNBND, BNDSQLSTT, and ENDBND along with any command data that the command can have. Individual READ calls at the application requester then receive the DDM object data stream structure response for each of the DDM commands.

Individual READ calls at the application server receive each DDM request data stream structure or command data stream structure. Individual WRITE calls at the server then transmit the DDM object data stream and reply data stream response structures for each of BGNBND, BNDSQLSTT, and ENDBND.

Refer to Chapter 4 on page 37 for further detail about DRDA DDM command sequences.

A bind flow is shown in Figure 13-3 on page 427.

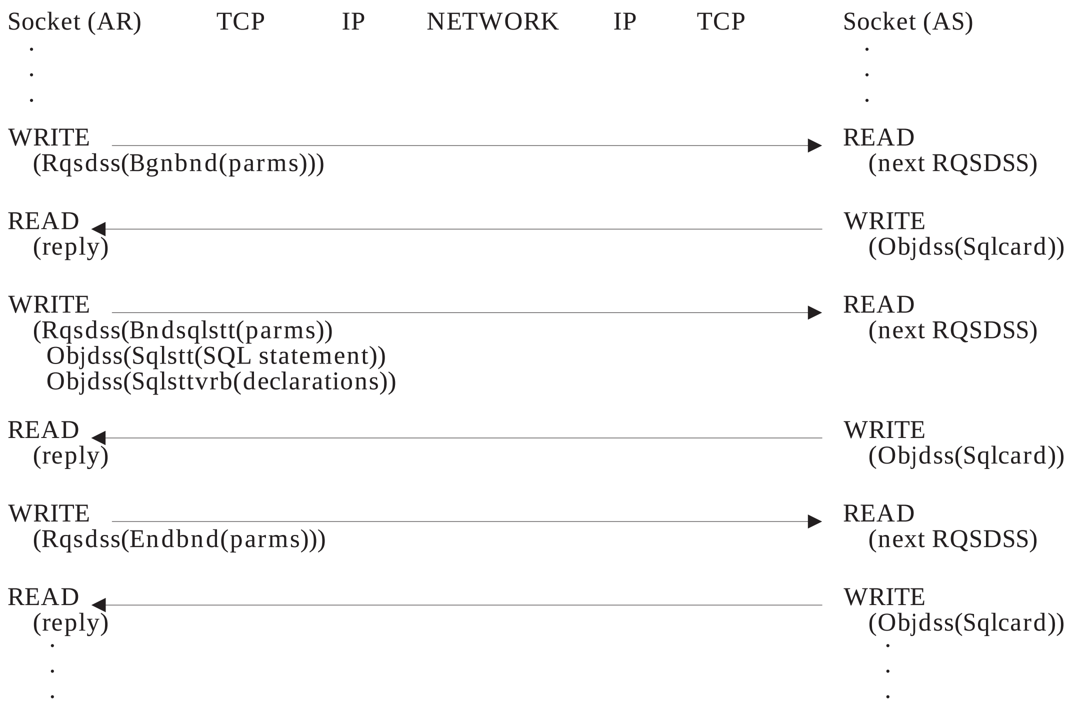


Figure 13-3 DRDA Bind Flows on TCP / IP

13.5.2.2 *SQL Statement Execution Flows*

Figure 13-4 on page 428 depicts DDM command processing using socket interface calls.

The physical flow of information consists of a sequence of packets containing DDM commands, FD:OCA data descriptors, FD:OCA data, and DDM reply messages.

Individual WRITE calls at the application requester transmit each of the DDM request data stream structures for OPNQRY and CNTQRY. Individual READ calls at the application requester then receive the DDM object data stream structure and reply message responses for the DDM commands.

Individual READ calls at the application server receive each DDM request data stream structure. Individual WRITE calls at the application server then transmit the DDM object data stream and reply message response structures for each of OPNQRY and CNTQRY.

Refer to Chapter 4 on page 37 for further detail about DRDA DDM command sequences.

Figure 13-4 on page 428 shows the SQL statement execution flow.

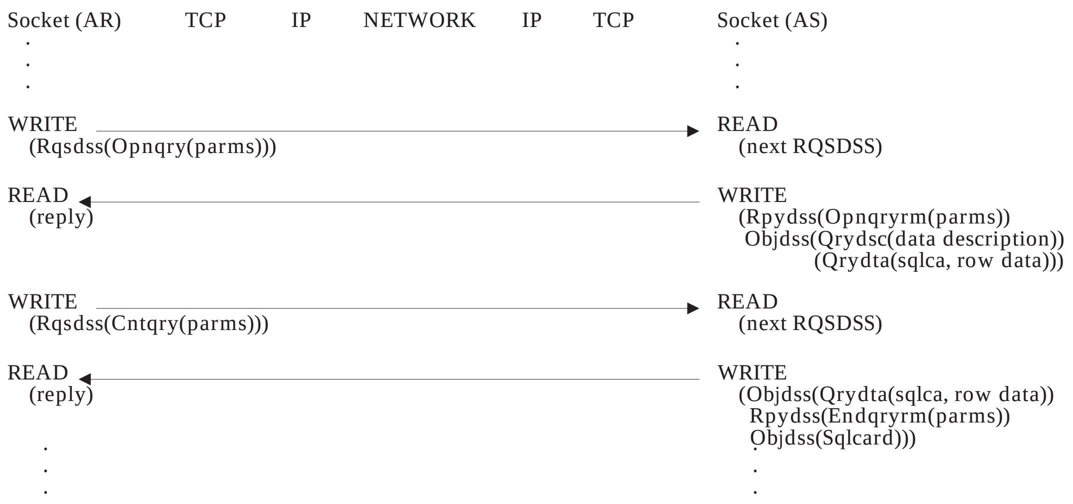


Figure 13-4 DRDA SQL Statement Execution Flows on TCP/IP

13.5.3 Terminating a Connection

Terminate connection processing closes a socket associated with the TCP/IP connection. Under normal circumstances, only an application requester initiates termination of the socket. In error situations, an application server can also initiate the termination of the socket.

The termination of the socket between an application requester and an instance of an application server terminates the communications between the application requester and that instance of the application server. The application server is also responsible to terminate the socket.

The application requester must ensure that all network connections associated with the execution of the application are terminated when the application normally or abnormally terminates.

On a TCP/IP connection, the application server receives an indication the socket is terminated. The termination includes an implied rollback at the application server. It is the responsibility of the application server to ensure a rollback during local termination processing at the application server.

The DDM Reference provides a general overview of the communications flows that make up a DRDA TCP/IP connection termination. The DDM term, TCPCMNT describes the termination of a TCP/IP connection. Figure 13-5 on page 429 shows the termination of a TCP/IP connection.

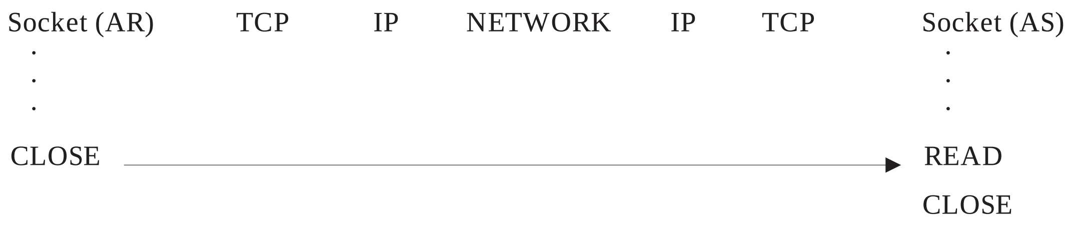


Figure 13-5 DRDA Termination Flows on TCP/IP

Figure 13-6 shows the abnormal termination of a TCP/IP connection. If the application server fails, the application server must attempt to return a permanent agent error reply message to provide diagnostics of the error to the application requester.

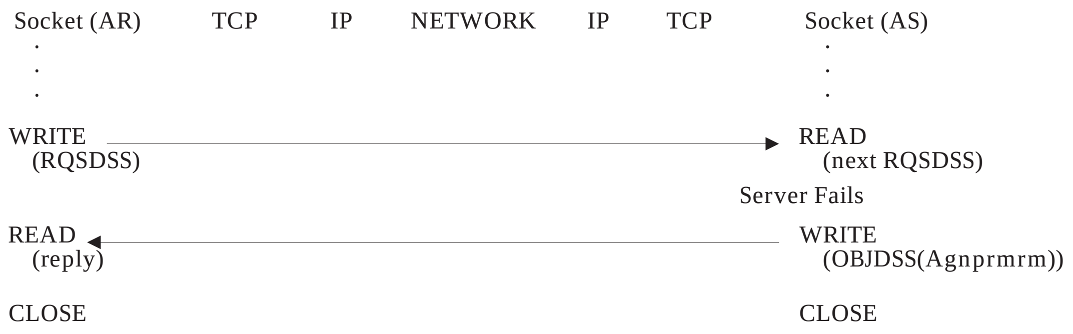


Figure 13-6 DRDA Server Abnormal Termination Flows on TCP/IP

13.5.4 Commit Flows

The physical flow of information for commit processing on TCP/IP connections consists of a sequence of packets containing DDM commands, and DDM reply messages.

13.5.4.1 Remote Unit of Work

Commit Flows: A WRITE call at the application requester transmits the DDM RDBCMM to commit the current unit of work. A READ call is issued to receive a response to the commit request.

A READ call at the application server receives the DDM request data stream structure. The RDB commits the unit of work. A WRITE call transmits the DDM ENDUOWRM, end unit of work, and an SQLCA identifying the resolution of the commit.

Refer to Chapter 4 on page 37 for further detail about DRDA DDM command sequences.

Figure 13-7 on page 430 shows the commit execution flow.

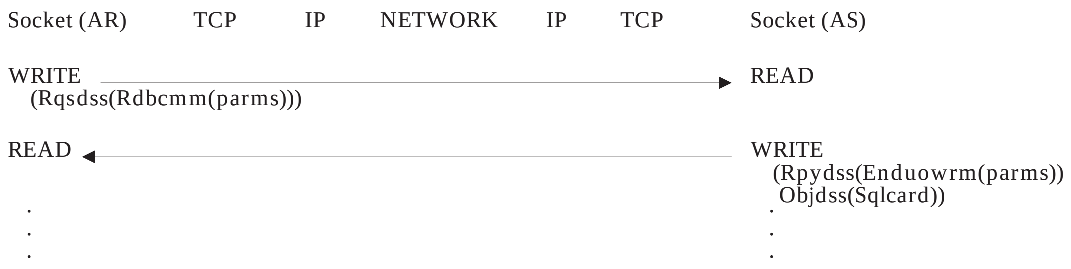


Figure 13-7 DRDA Commit Flows on TCP/IP

Rollback Flows: The physical flow of information for rollback processing on TCP/IP connections is the same as the commit flows on TCP/IP connections. See Figure 13-7 and replace RDBCMM with RDBRLLBCK.

13.5.4.2 *Distributed Unit of Work Using DDM Sync Point Manager*

Commit Flows: The application requester invokes the DDM Sync Point Manager to coordinate the commit.

Refer to DDM SYNCPTOV term for a definition of the DRDA Level 3 2-phase command sequences and logging requirements. Figure 13-8 on page 431 shows the four message two-phase commit with each application server that participated in the current unit of work. Prior to starting any units of work the application requester exchanges log information with the application server. The log information is used if the commit operation fails and resynchronization is required to complete the commit operation. When initiating a unit of work with an application server, the sync point manager at application requester issues a WRITE call to transmit the new unit of work identifier sync point control request to the sync point manager at the application server. No reply is expected from the application server.

To initiate the commit, the application requester sync point manager issues a WRITE call to transmit the prepare to commit sync point control request to the application server. A READ call is then issued to receive the reply from the application server's sync point manager.

A READ call at the application server receives the prepare to commit sync control request. After the RDB has prepared to commit, the sync point manager sends a request that the unit of work is ready to be committed by issuing a WRITE call with a request to commit sync control reply data back to the application requester. Another READ is issued to receive the outcome of the commit.

At the application requester, the READ completes with the request to commit sync control reply data. The sync point manager commits the unit of work and issues a WRITE call with the committed sync control request to the application server specifying implicit or explicit forget processing. Implicit forget processing is a performance option to save a network message and improve overall commit performance. Another READ is issued to receive the outcome of the commit at the application server.

The READ completes at the application server with the committed sync control request. The sync point manager commits and forgets the unit of work. An optional WRITE call transmits the forget sync control reply data to the application requester. Otherwise the next successful reply infers the forget.

A READ call at the application requester receives the explicit forget or an implied forget. The unit of work is forgotten and control is returned to the application requester and then to the application.

Figure 13-8 shows the two-phase commit execution flow.

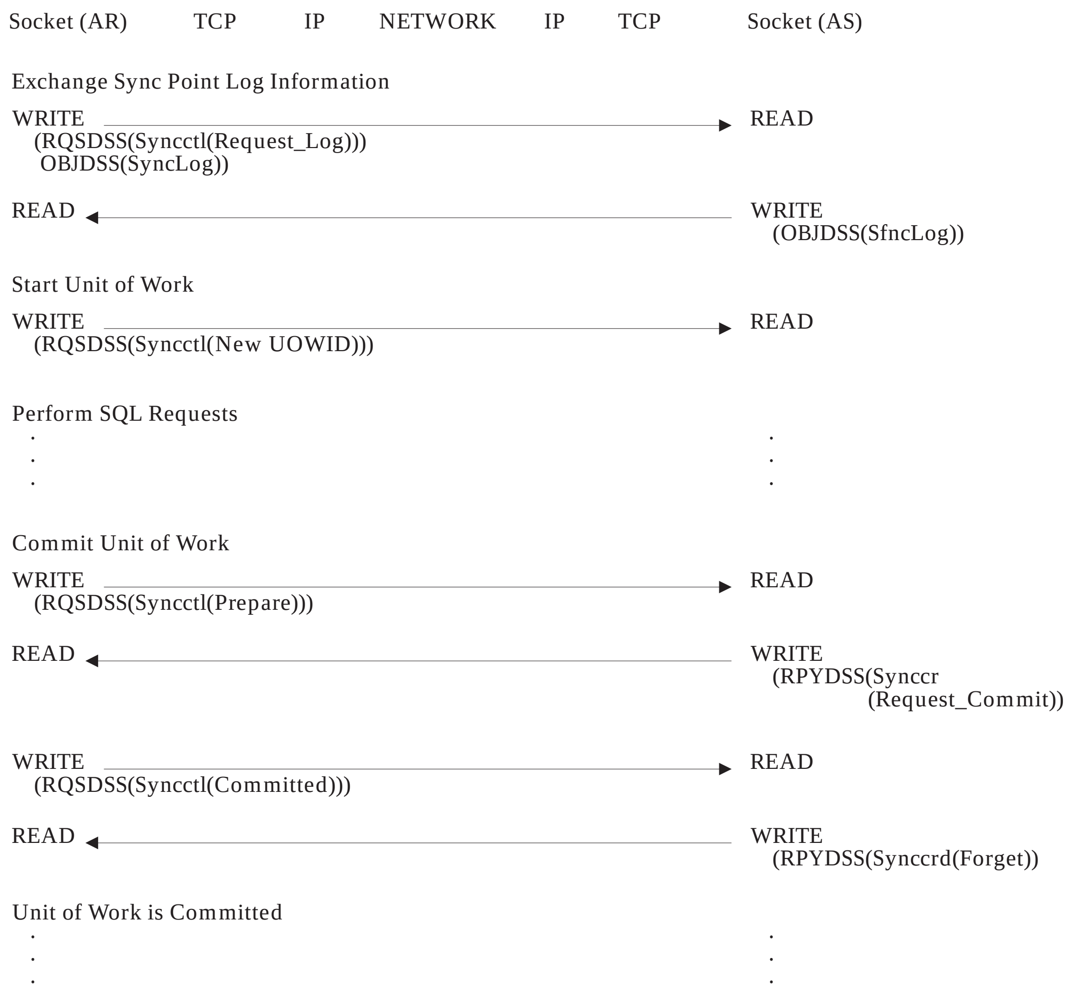


Figure 13-8 TCP/IP Distributed Unit of Work Commit Flow

Rollback Flows: The application requester invokes the DDM Sync Point Manager to coordinate the rollback. Figure 13-9 on page 432 shows the one message rollback execution flow. The sync point manager rollbacks the unit of work and issues a WRITE call with the rollback sync control data stream structure to the application server.

A READ call at the application server receives the rollback sync control data stream structure. The sync point manager rollbacks and forgets the unit of work.

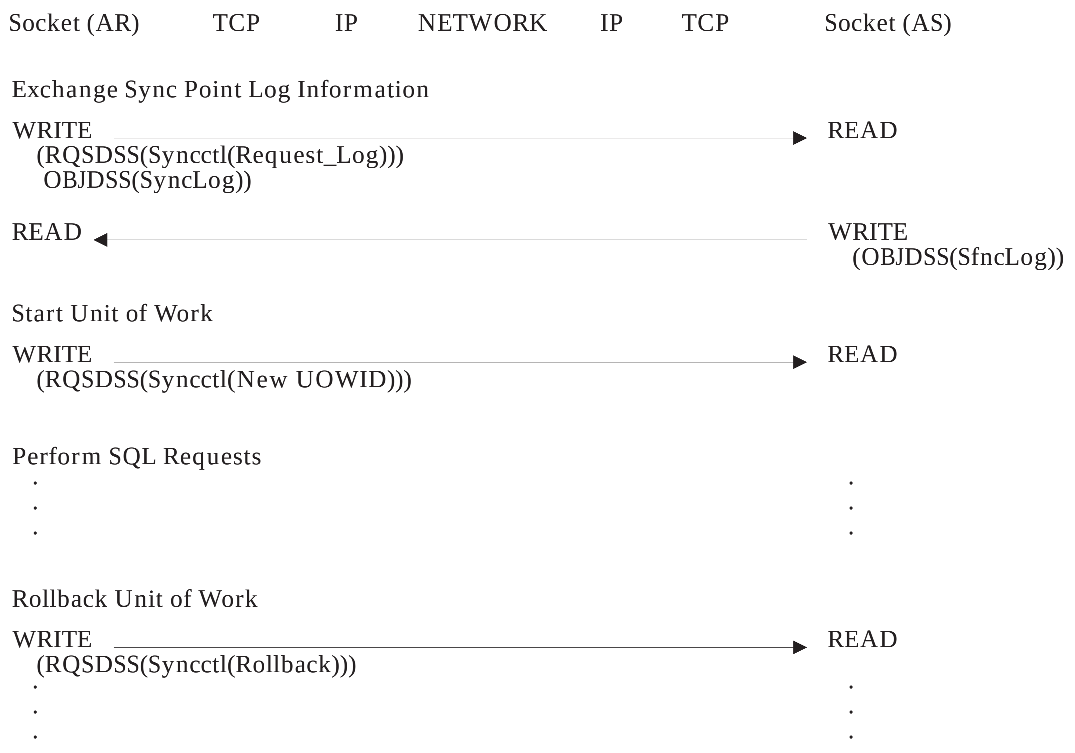


Figure 13-9 TCP/IP Distributed Unit of Work Rollback Flow

13.5.5 Handling Connection Failures

There are facilities available in TCP/IP to allow the application requester and the instance of the application server to be informed if the TCP/IP connection linking the application requester to the instance of the application server fails. The application server must then implicitly roll back the effects of the application and deallocate all database management resources supporting the application.

Distributed unit of work connections, in the case of a failure on a TCP/IP connection, the application requester is responsible for rolling back all other resources involved in the unit of work, which might include initiating backout processing to a sync point manager to backout the application servers on protected network connections.

After all resources are rolled back, the application requester must report the failure to the application in the SQLCA. The application requester can then take one of two actions:

- Reject any subsequent SQL request from the application.
- Treat the next SQL request from the application as the beginning of a new unit of work. In this case, it would begin the DRDA initialization sequence again.

13.6 TCP/IP Environment Usage in DRDA

This section describes considerations for problem determination in TCP/IP environments, and rules usage and target program names usage in TCP/IP environments.

13.6.1 Problem Determination in TCP/IP Environments

The DRDA environment involves remote access to relational database management systems. Because the access is remote, enhancements to the local problem determination process were needed. These enhancements use Network Management tools and techniques. These tools and techniques are:

- Standard Focal Point Messages
- Focal Point support
- Correlation and Correlation display
- Data Collection

13.6.1.1 Standard Focal Point Messages

The commonly accepted focal point messages in the TCP/IP environment are Simple Network Management Protocols (SNMP) traps. At this time, DRDA does not define SNMP traps.

13.6.1.2 Focal Point Support

DRDA assumes a focal point is available in a TCP/IP environment and assumes the use of SNA alerts.

13.6.1.3 Correlation and Correlation Display

Correlation values that are generated in a TCP/IP environment have the following format:

```
x.yz
where:
x  8-byte character representation of the 4-byte IP address
   of the application requester
.  delimiter
y  4-byte character representation of the 2-byte socket address
   of the application requester
z  6-byte binary value (possibly a clock value) that makes
   the correlation value unique
```

The specific rules for the display of a correlation value generated are:

1. Display the correlation token in the format x.y.z.
2. Display the x.y portion of the correlation token as character data in x.y format (13 bytes).
3. Display the z part of the correlation value as a string of hexadecimal characters (12 bytes). A period is used to delimit the x.y from z.

Correlation Between Focal Point Messages and Supporting Data

Correlation between focal point messages and supporting data at each location, as well as cross-location, is done through correlation tokens. The correlation token is the ACCRDB *crrtkn* parameter value. The *crrtkn* value can be inherited at the application requester from the operating environment. If the inherited value matches the format of a DRDA-defined correlation token (x.yz), then it is sent at ACCRDB in the *crrtkn* parameter. If the application requester does not inherit a correlation value, or the value does not match the format of a DRDA-defined correlation token, then the application requester must generate a correlation token. The correlation value is required in focal point messages and supporting diagnostic information.

13.6.2 Rules Usage for TCP/IP Environments

This section consists of the TCP/IP usage of the rules defined in Chapter 7 on page 281.

13.6.2.1 TCP/IP Usage of Connection Allocation Rules

CA2 Usage

Connections between an application requester and an application server must have the following socket options:

- SO_KEEPALIVE: keep connection alive to provide timely detection of a broken connection.
- SO_LINGER: linger on close if data present to allow detection of a broken connection.

CA3 Usage

A connection between an application requester and an application server using remote unit of work protocols must use the SYNCPTMGR at Level 0.

A connection between an application requester and an application server using distributed unit of work protocols can have SYNCPTMGR at Level 0 or SYNCPTMGR at Level 5. If either the application requester or application server does not support SYNCPTMGR at Level 5, the connection must use SYNCPTMGR at Level 0.

CA12 Usage

An application requester operating using distributed unit of work protocols can initiate a TCP/IP connection with one or more application servers in a unit of work.

13.6.2.2 TCP/IP Usage of Commit/Rollback Processing Rules

CR2 Usage

Remote unit of work application servers or distributed unit of work application servers on connections using SYNCPTMGR at Level 0 must inform the application requester when the current unit of work at the application server ends as a result of a commit or rollback request by an application or application requester request (dynamic commit and dynamic rollback are not allowed in a distributed unit of work connection). This information is returned in the RPYDSS, containing the ENDUOWRM reply message. This RPYDSS is followed by an OBJDSS containing an SQLCARD with information that is input to the SQLCA to be returned to the application. If multiple commit or rollbacks occur prior to exiting a stored procedure, only one ENDUOWRM is returned. See rule CR13 in Section 7.4 on page 284 for setting the *uowdsp* parameter when multiple commit and/or rollbacks occur in a stored procedure. See CR6 for the SQLSTATEs to return.

13.6.2.3 TCP/IP Usage of Security (SE Rules)

SE2 The application server must support SECMGR Level 5 to be able to obtain the verified end-user name associated with the TCP/IP connection. DRDA Level 3, therefore, requires one of the following mechanisms:

- The specification of one of the following DRDA supported security mechanisms:
  - — DCE-based security mechanism for end-user identification and authentication
  - — Userid and password (DDM *usridpwd*) security mechanism for end-user identification and authentication
  - — Userid and new password (DDM *usridnwpwd*) security mechanism for end-user identification, authentication, and the changing of the password
  - — Userid only (DDM *usridonl*) security mechanism for end-user identification

  ACCRDB must be rejected with MGRDEPRM if the application server does not obtain the verified end-user name.

13.6.2.4 TCP/IP Usage of Serviceability Rules

SV1 Usage
The application requester must generate diagnostic information and may generate a focal point message when the TCP/IP connection to the application server ends unexpectedly.

SV8 Usage
The DDM UOWID or the *crrtkn* on the ACCRDB must be present in the alert, in the supporting data information, or in diagnostic information.

13.6.2.5 TCP/IP Usage of Relational Database Names Rules

RN2 Usage
DRDA associates an RDB_NAME with a specific port at a unique IP address. DRDA, however, does not define the mechanism that derives the IP address and port pair from the RDB_NAME. The particular derivation mechanisms are specific to the environment.

It is the responsibility of the application requester to determine the RDB_NAME name of the relational database and to map this name to an IP address and port.

RN3 Usage
More than one RDB_NAME may exist for a single IP address. An RDB_NAME must map to an IP address and port.

RN4 Usage
DRDA permits the association of more than one RDB_NAME with a single port at an IP address.

13.6.2.6 TCP/IP Usage of PORT for DRDA Service Rules

TPN1 Usage
The PORT identifying DRDA application servers and database servers must support the registered TCP/IP well known port for a DRDA application server or any non-registered TCP/IP port.

TPN2 DRDA allows DDM file servers and DRDA SQL servers to use either the same well known port or different well known port.

TPN3 Usage
Registered TCP/IP well known port for a DRDA application server is 446.

TPN4 Usage
Multiple ports for a DRDA application server might exist for a single IP address.

TPN5 Usage
A well known port for an application server is unique for an IP address.

TPN6 Usage
A well known port for a DRDA application server must provide all the capabilities that DRDA requires.

TPN7 Usage
The well known port that provide DRDA capabilities may perform additional non-DRDA work. These ports are not required to perform additional non-DRDA.

TPN8 Usage
The DRDA well known port must be supported at each IP address with at least one application server providing DRDA capabilities.

13.6.3 Service Names

TCP/IP requires that an application requester specify the port of the application server when initiating a connection. The application requester determines the port of the application server during the process of resolving the RDB_NAME of the application server to an IP address. DRDA allows the use of any valid port that meets the standards of the TCP/IP architecture and that the application server supports.

To avoid potential name conflicts, the application server port should be, but need not be, a registered TCP/IP well known port for a DRDA application server. This well known port is 446.

The default DRDA well known port for an application server is 446. The default well known port must be supported at each IP address that has an application server providing DRDA capabilities. An application requester can then assume the existence of a well known port 446 at any IP address providing DRDA capabilities, and default to port 446 when a request requiring a TCP/IP connection does not specify a port.

Appendix A

Building Statement-Level SQLCAs for Multi-Row Fetches

Building the Statement-Level SQLCAs for Multi-Row Fetch Operations

Table A-1 and the text that follows the table describes the building of the statement-level SQLCAs for multi-row fetch operations. The text also defines when the SQLCA should be returned to the application.

The actual process and data in the SQLCAs can vary by product, so the individual product documentation should be consulted for the exact process and format.

Multi-row fetch is not supported in DRDA Level 1.

Table A-1 Setting of the Statement-Level SQLCA

Condition	Action
• No errors • No warnings • All requested rows returned	• SQLCODE = Execution of SQL statement was successful • SQLSTATE = ’00000’ • SQLERRD3 = Number of rows • SQLERRD4 = Number of bytes used for each result row • SQLERRD5 = 0
• No errors • No warnings • Fewer than all requested rows returned, due to EOF	• SQLCODE = Execution of SQL statement was successful • SQLSTATE = ’00000’ • SQLERRD3 = Number of rows returned • SQLERRD4 = Number of bytes used for each result row • SQLERRD5 = SQLCODE value for position of the cursor is after the last row of the result table If diagnostic area SQLCAs are provided, then the diagnostic area SQLCA after the SQLCA for the last good row will contain SQLSTATE X’02000’ and the SQLCODE value for the position of the cursor is after the last row of the result table
• EOF • No errors • No warnings	• SQLCODE = value for position of the cursor is after the last row of the result table • SQLSTATE=’02000’ • SQLERRD3 = 0

Condition	Action
	• SQLERRD4 = Number of bytes used for each result row • SQLERRD5 = 0
• No errors • Warning Flags • All requested rows returned	• SQLCODE = Execution of SQL statement was successful • SQLSTATE = SQLSTATE of last Warning • SQLWARN0 = ’W’ • SQLWARNx = ’W’ (The warning flags are an accumulation of all warning flags set while processing the multi-row fetch.) • SQLERRD3 = Number of rows returned • SQLERRD4 = Number of bytes used for each result row • SQLERRD5 = 0
• No errors • Positive SQLCODE • All requested rows returned	• SQLCODE = SQLCODE of last positive SQLCODE • SQLSTATE = SQLSTATE equivalent to the SQLCODE • SQLERRD3 = Number of rows returned • SQLERRD4 = Number of bytes used for each result row • SQLERRD5 = 0
• No errors • Warning Flags • Positive SQLCODE • All requested rows returned	• SQLCODE = SQLCODE of last positive SQLCODE • SQLSTATE = SQLSTATE equivalent to the SQLCODE • SQLWARN0 = ’W’ • SQLWARNx = ’W’ (The warning flags are an accumulation of all warning flags set while processing the multi-row fetch.) • SQLERRD3 = Number of rows returned • SQLERRD4 = Number of bytes used for each result row • SQLERRD5 = 0
• AS detected error (Assume FETCH NEXT)	1. FETCH #1 - Return the good rows. • SQLCODE = Execution of SQL statement was successful • SQLSTATE=’00000’ • SQLERRD3 = Number of rows returned

Condition	Action
	• SQLERRD4 = Number of bytes used for each result row • SQLERRD5 = 0 2. FETCH #2 - Return the error. • SQLCODE = error • SQLSTATE = SQLSTATE equivalent to the SQLCODE • SQLERRD3 = 0 • SQLERRD5 = 0
• AR detected error - 22002, 22001, 42806, 22021, or 55021.	1. FETCH #1 - Return the good rows. • SQLCODE = Execution of SQL statement was successful • SQLSTATE='00000' • SQLERRD3 = Number of rows returned • SQLERRD4 = Number of bytes used for each result row • SQLERRD5 = 0 2. What happens next depends on the type of FETCH: • If FETCH #2 is FETCH FIRST/BEFORE/LAST/AFTER/ABSOLUTE, no error. • For any other type of FETCH then: — SQLCODE = Value for cursor position is unknown — SQLSTATE = '24513' — SQLERRD3 = 0 — SQLERRD5 = 0

SQLERRD3 will contain the number of records actually fetched, and the SQLWARN flags will be set if they were set during any single fetch operation. SQLERRD3 will also be set for single-row fetches. SQLERRD3 is set at the application requester.

For multi-row fetches, SQLERRD4 will contain the number of bytes used for each result row. This is the maximum row size. SQLERRD4 is set at the application requester.

For multi-row fetches, if the number of rows returned is less than the number of rows requested on the FOR n ROWS clause due to reaching the end of cursor, then SQLERRD5 will contain an SQLCODE value that indicates the position of the cursor after the last row of the result table.

The SQLCA stores errors and warnings found during the multi-row fetch. If an error results in an SQLSTATE greater than X'02999' or an end-of-file condition, all good rows fetched before the error or end-of-file are returned to the SQL application with SQLSTATE X'00000'. The error or end-of-file is not returned until the next fetch attempt, unless changes have been made to the table which would cause another good row to be fetched. Consider the following examples, where we attempt to fetch 10 rows with a multi-row fetch statement.

- An error condition is detected on the fifth row that would result in an SQLSTATE greater than X‘02999’. SQLERRD3 will be set to 4 for the four returned rows, and the SQLSTATE will be set to X‘00000’. The SQL application will get the error on the next fetch.
- An end-of-file condition is detected on the fifth row. SQLERRD3 will be set to 4 for the four returned rows, and the SQLSTATE will be set to X‘00000’. SQLERRD5 will be set to indicate the end-of-file condition. The SQL application will get the end of file (SQLSTATE X‘02000’) on the next fetch.
- An end-of-file condition is detected on the fifth row. SQLERRD3 will be set to 4 for the four returned rows, and the SQLSTATE will be set to X‘00000’. SQLERRD5 will be set to indicate the end-of-file condition. After the FETCH, 1 more record is inserted into the table. On the next FETCH call, an end-of-file condition is detected after the new row has been fetched. SQLERRD3 will be set to 1 for the one returned row, and the SQLSTATE will be set to X‘00000’. SQLERRD5 will be again set to the end-of-file condition. The SQL application will get the end-of-file (SQLSTATE X‘02000’) on the next fetch.

For data mapping errors, if indicator variables are provided, return all rows to the SQL application, marking the errors in the indicator variables. The SQLSTATE will contain the warning from the last data mapping error, and the cursor will point at the last row fetched. If only some or no indicator variables are provided, return all rows as above until the first data mapping error is detected which does not have indicator variables. Return the good rows and set the SQLSTATE and SQLCODE to reflect the execution for the returned rows. If the next fetch is FIRST, BEFORE, AFTER, LAST, or ABSOLUTE, then reposition the cursor and no error is returned. If the next fetch is NEXT, CURRENT, RELATIVE, or PRIOR, then return SQLSTATE X‘24513’ and an SQLCODE that indicates the value for the cursor position is unknown. The SQLERRM contains the SQLSTATE and SQLCODE of the error that caused the cursor to be placed in an unknown position.

Consider the following examples where an attempt is made to fetch 10 rows:

- The host structure array provided indicator variables for every element of the structure and the fifth and seventh rows have data mapping errors in them. The sixth row also has a truncation warning. SQLERRD3 will be set to 10 for the ten returned rows, and the SQLSTATE will be set to the warning from the seventh row fetched. The indicator variables for the fifth and seventh row will indicate data mapping errors were found. SQLWARN1 will be set to W to signal that a truncation occurred. The cursor will be on the tenth row.
- Structure array elements 1 through 5 have indicator variables for them, and structure elements 6 through 10 do not. The fifth and seventh rows have data mapping errors in them. SQLERRD3 will be set to 6 for the six returned rows, and the SQLSTATE will be set to the warning from the fifth row fetched. The indicator variable for the fifth row will indicate a data mapping error was found. The error will be returned as indicated in Table A-1 on page 437.

DDM Managers, Commands, and Reply Messages

This appendix is provided to help an implementer sort out what level of DDM managers are required to support a specified level of DRDA, and also contains a summary of the required and optional DDM commands and replies as they relate to each level of DRDA.

Section B.1 shows the relationship of types of distribution (Remote Unit of Work and Distributed Unit of Work) to the DDM managers. Section B.2 on page 442 defines the DDM commands, replies, and parameters in relationship to the DDM manager and in relationship to the DRDA level.

B.1 DDM Manager Relationship to DRDA Functions

The following table associates the DDM managers with the specified DRDA types of distribution. In some cases, the DDM level in the table is not specific; for example, "0 or 3". In those cases, the DRDA level does not require a specific DDM manager level, but is dependent on the level of function required and the level of manager required to support that function. For example, if the product wants to implement all the recent DRDA Level functions on a DRDA Remote Unit of Work base while using a TCP/IP network protocol, the product would build an SQLAM Level 3 and CMNTCPIP Level 5 and would not build CMNAPPC, CMNSYNCPT, and SYNCPTMGR support.

Table B-1 DDM Manager Relationship to DRDA Level

Manager	DRDA Remote Unit of Work	DRDA Distributed Unit of Work	DRDA Level 3
AGENT	3	3 or 4	3, 4, or 5
CCSID	0 or ccsid#	0 or ccsid#	0 or ccsid#
CMNAPPC	0 or 3	3	0 or 3
CMNSYNCPT	0 or 4	4	0 or 4
CMNTCPIP	0 or 5	0 or 5	0 or 5
RDB	3	3	3
SQLAM	3	4	3, 4, or 5
SYNCPTMGR	0, 4, or 5	4 or 5	0, 4, or 5
SECMGR	1	1	5

B.2 DDM Commands and Reply Messages

This section contains tables that associate DDM commands and replies in relationship to a specific DDM manager level. The tables also associates the parameters for the commands and replies in relationship to the DRDA levels.

The terms required or optional follow the definitions outlined in the DDM architecture for REQUIRED and OPTIONAL. In some cases, we further qualify the item as conditional, ignorable, mutually inclusive, mutually exclusive, or dependent. If it is Conditional, then there are extra conditions placed on the term through DRDA or DDM. If it is Ignorable, Mutually Inclusive, or Mutually Exclusive, the extra conditions are described in DDM. If it is Dependent, then this parameter might be required dependent on the level of another manager that is optional for this level of DRDA.

In some cases, DRDA overrides the optionality of the term. For example, the *extnam* instance variable is optional in DDM but is required in DRDA. The requirement or optionality of a term is shown in the following tables and includes the DRDA overrides.

The semantics of the support in the application requester and application server for required and optional commands, replies, and data objects are described in the SUBSETS term in DDM. Further overriding conditions are described in Section 7.9 on page 295.

If an item is listed as not defined, it is because the item is not defined for the designated level of DRDA.

The ABNUOWRM Reply Message

Table B-2 ABNUOWRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Required	Required	Required
Rdbnam (relational database name)	Required	Required	Required
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The ACCRDB Command

Table B-3 ACCRDB Command Instance Variables

Instance Variable	**SQLAM Level 3**	**SQLAM Level 4**	**SQLAM Level 5**
Rdbacccl (access manager class)	Required	Required	Required
Crrtkn (correlation token)	Optional	Required	Required
Rdbnam (name of remote database)	Required	Required	Required
Prdid (product-specific identifier)	Required	Required	Required
Typdefnam (data type definition name)	Required	Required	Required
Typdefovr (data type definition override)	Required	Required	Required
Rdbalwupd (rdb to allow updates)	Optional	Optional	Optional
Prddta (product-specific data)	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable
Sttdecdel (decimal delimiter)	Optional	Optional	Optional
Sttstrdel (string delimiter)	Optional	Optional	Optional
Trgdftrt (target default values return)	Optional	Optional	Optional

Table B-4 Reply Objects for the ACCRDB Command

Reply Object	**SQLAM Level 3**	**SQLAM Level 4**	**SQLAM Level 5**
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqlcard (SQLCA reply data)	Optional	Optional	Optional

The ACCRDBRM Reply Message

Table B-5 ACCRDBRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Required	Required	Required
Prdid (product identifier)	Required	Required	Required
Typdefnam (data type definition name)	Required	Required	Required
Typdefovr (TYPDEF overrides)	Required	Required	Required
Rdbinttkn (RDB interrupt token)	Optional	Optional	Optional
Crrtkn (correlation token)	Required / Conditional	Required / Conditional	Required / Conditional
Srvdgn (server diagnostic information)	Optional	Optional	Optional
Pkgdftcst (package default character subtype)	Optional / Conditional	Optional / Conditional	Optional / Conditional
Srvlst (target server list)	Not defined	Not defined	Optional / Ignorable
Userid (user ID at the target system)	Optional / Conditional	Optional / Conditional	Optional / Conditional

The ACCSEC Command

Table B-6 ACCSEC Command Instance Variables

Instance Variable	**SECMGR Level 5**
Secmgrnm (security manager name)	Optional / Ignorable
Secmec (security mechanism)	Required

Table B-7 ACCSECRD Reply Object Instance Variables

Instance Variable	**SECMGR Level 5**
Secmec (security mechanism)	Required

The AGNPRMRM Reply Message

Table B-8 AGNPRMRM Reply Message Instance Variables

Instance Variable	**SQLAM Level 3**	**SQLAM Level 4**	**SQLAM Level 5**
Svrcod (severity code)	Required	Required	Required
Rdbnam (relational database name)	Optional	Optional	Optional
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The BGNBND Command

Table B-9 BGNBND Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Rdbnam (name of remote database as in ACCRDB)	Optional	Optional	Optional
Pkgnamct (package name and consistency token)	Required	Required	Required
Vrsnam (package version name)	Optional	Optional	Optional
Pkgrplvrs (replaced package version name)	Optional	Optional	Optional
Bndchkexs (bind existence checking)	Optional	Optional	Optional
Bndcrtctl (bind creation control)	Optional	Optional	Optional
Bndexpopt (bind explain option)	Optional	Optional	Optional
Decprc (decimal precision)	Optional	Optional	Optional
Dftrdbcol (default RDB collection identifier)	Optional	Optional	Optional
Pkgathopt (package authorization option)	Optional	Optional	Optional
Pkgdftcc (package default CCSID)	Optional	Optional	Optional
Pkgdftcst (default character subtype)	Optional	Optional	Optional
Pkgisolvl (package isolation level)	Required	Required	Required
Pkgrplopt (package replacement option)	Optional	Optional	Optional
Pkgownid (package owner identifier)	Optional	Optional	Optional
Qryblkctl (query block protocol control)	Optional	Optional	Optional
Rdbrlsopt (RDB release option)	Optional	Optional	Optional
Sttdatfmt (date format of statement)	Optional	Optional	Optional
Sttdecdel (statement decimal delimiter)	Optional	Optional	Optional
Sttstrdel (statement string delimiter)	Optional	Optional	Optional
Stttimfmt (time format of statement)	Optional	Optional	Optional
Title (brief description of package)	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable
Dgrioprl (degree of I/O Parallelism)	Not defined	Optional/ Ignorable	Optional/ Ignorable
Pkgathrul (package authorization rules)	Not defined	Not defined	Optional/ Ignorable

Table B-10 Command Objects for the BGNBND Command

Command Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Bndopt (bind option)	Not defined	Not defined	Optional

Table B-11 Reply Objects for the BGNBND Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqlcard (SQLCA reply data)	Required	Required	Required

The BGNBNDRM Reply Message

Table B-12 BGNBNDRM Reply Message Instance Variables

Instance Variable	**SQLAM Level 3**	**SQLAM Level 4**	**SQLAM Level 5**
Svrcod (severity code)	Required	Required	Required
Rdbnam (relational database name)	Required	Required	Required
Pkgnamct (rdb package name and consistency token)	Required	Required	Required
Vrsnam (version name)	Required	Required	Required
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The BNDSQLSTT Command

Table B-13 BNDSQLSTT Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Rdbnam (name of remote database as in ACCRDB)	Optional	Optional	Optional
Pkgnamcsn (package name, consistency token and section number)	Required	Required	Required
Sqlsttnbr (source application statement number)	Optional	Optional	Optional
Bndsttasm (bind statement assumptions)	Optional	Optional	Optional

Table B-14 Command Objects for the BNDSQLSTT Command

Command Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqlstt (SQL statement to be bound in the application server package)	Required	Required	Required
Sqlsttvrb (description of each variable)	Optional	Optional	Optional

Table B-15 Reply Objects for the BNDSQLSTT Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqlcard (SQLCA reply data)	Required	Required	Required

The CLSQRY Command

Table B-16 CLSQRY Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Rdbnam (name of remote database as in ACCRDB)	Optional	Optional	Optional
Pkgnamcsn	Required	Required	Required

Table B-17 Reply Objects for the CLSQRY Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqlcard (SQLCA reply data)	Required	Required	Required

The CMDATHRM Reply Message

Table B-18 CMDATHRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Required	Required	Required
Rdbnam (relational database name)	Optional	Optional	Optional
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The CMDCHKRM Reply Message

Table B-19 CMDCHKRM Reply Message Instance Variables

Instance Variable	**SQLAM Level 3**	**SQLAM Level 4**	**SQLAM Level 5**
Svrcod (severity code)	Required	Required	Required
Rdbnam (relational database name)	Optional	Optional	Optional
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The CMDNSPRM Reply Message

Table B-20 CMDNSPRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Required	Required	Required
Codpnt (code point attribute)	Required	Required	Required
Rdbnam (relational database name)	Optional	Optional	Optional
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The CMDVTLRM Reply Message

Table B-21 CMDVLTRM Reply Message Instance Variables

Instance Variable	**SQLAM Level 3**	**SQLAM Level 4**	**SQLAM Level 5**
Svrcod (severity code)	Not defined	Required	Required
Rdbnam (relational database name)	Not defined	Required	Required
Srvdgn (server diagnostic information)	Not defined	Optional	Optional

The CMMRQSRM Reply Message

Table B-22 CMMRQSRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Not defined	Required	Required
Rdbnam (relational database name)	Not defined	Required	Required
Cmmtyp (commitment request type)	Not defined	Required	Required
Srvdgn (server diagnostic information)	Not defined	Optional	Optional

The CNTQRY Command

Table B-23 CNTQRY Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Rdbnam (name of remote database as in ACCRDB)	Optional	Optional	Optional
Pkgnamcsn (package name, consistency token and section number)	Required	Required	Required
Qryblksz (query block size)	Required	Required	Required
Qryrelscr (query relative scrolling action)	Not defined	Optional	Optional
Qryrownbr (query row number)	Not defined	Optional	Optional
Qryrfrtbl (query refresh answer set table)	Not defined	Optional	Optional
Nbrrow (number of fetch rows)	Not defined	Optional	Dependent on SQLAM level
Maxblkext (maximum number of extra blocks)	Not defined	Not defined	Optional

Table B-24 Reply Objects for the CNTQRY Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqlcard (SQLCA reply data)	Required / Conditional	Required / Conditional	Required / Conditional
Qrydta (query answer set data)	Optional	Optional	Optional

The DRPPKG Command

Table B-25 DRPPKG Command Instance Variables

Instance Variable	**SQLAM Level 3**	**SQLAM Level 4**	**SQLAM Level 5**
Rdbnam (name of remote database as in ACCRDB)	Optional	Optional	Optional
Pkgnam (package grouping name and identifier)	Required	Required	Required
Vrsnam (version name)	Optional	Optional	Optional

Table B-26 Reply Objects for the DRPPKG Command

Reply Object	**SQLAM Level 3**	**SQLAM Level 4**	**SQLAM Level 5**
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqlcard (SQLCA reply data)	Required	Required	Required

The DSCINVRM Reply Message

Table B-27 DSCINVRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Required	Required	Required
Dscerrcd (description error code)	Required	Required	Required
Rdbnam (relational database name)	Required	Required	Required
Fdodsc (FD:OCA data descriptor)	Required	Required	Required
Fdodscoff (FD:OCA descriptor offset)	Required	Required	Required
Fdotrpoff (FD:OCA triplet offset)	Required	Required	Required
Fdoprmoff (FD:OCA triplet parameter offset)	Required	Required	Required
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The DSCPVL Command

Table B-28 DSCPVL Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Rdbnam (name of remote database as in ACCRDB)	Optional	Optional	Optional
Sqlobjtyp (SQL object type)	Required	Required	Required

Table B-29 Command Objects for the DSCPVL Command

Command Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqlobjnam (SQL object name)	Required	Required	Required

Table B-30 Reply Objects for the DSCPVL Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqlpard (SQLPA reply data)	Required / Mutually Exclusive	Required / Mutually Exclusive	Required / Mutually Exclusive
Sqlcard (SQLCA reply data)	Required / Mutually Exclusive / Conditional	Required / Mutually Exclusive / Conditional	Required / Mutually Exclusive / Conditional

The DSCRDBTBL Command

Table B-31 DSCRDBTBL Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Rdbnam (name of remote database as in ACCRDB)	Optional	Optional	Optional

Table B-32 Command Objects for the DSCRDBTBL Command

Command Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqlobjnam (SQL object name)	Required	Required	Required

Table B-33 Reply Objects for the DSCRDBTBL Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqldard (SQLDA reply data)	Required / Mutually Exclusive	Required / Mutually Exclusive	Required / Mutually Exclusive
Sqlcard (SQLCA reply data)	Required / Mutually Exclusive / Conditional	Required / Mutually Exclusive / Conditional	Required / Mutually Exclusive / Conditional

The DSCSQLSTT Command

Table B-34 DSCSQLSTT Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6
Rdbnam (name of remote database as in ACCRDB)	Optional	Optional	Optional	Optional
Pkgnamcsn (package name, consistency token and section number)	Required	Required	Required	Required
Typsqlda (input | output)	—	—	—	Optional

Table B-35 Reply Objects for the DSCSQLSTT Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqldard (SQLDA reply data)	Required / Mutually Exclusive	Required / Mutually Exclusive	Required / Mutually Exclusive
Sqlcard (SQLCA reply data)	Required / Mutually Exclusive / Conditional	Required / Mutually Exclusive / Conditional	Required / Mutually Exclusive / Conditional

The DTAMCHRM Reply Message

Table B-36 DTAMCHRM Reply Message Instance Variables

Instance Variable	**SQLAM Level 3**	**SQLAM Level 4**	**SQLAM Level 5**
Svrcod (severity code)	Required	Required	Required
Rdbnam (relational database name)	Required	Required	Required
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The ENDBND Command

Table B-37 ENDBND Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Rdbnam (name of remote database as in ACCRDB)	Optional	Optional	Optional
Pkgnamct (package name and consistency token)	Required	Required	Required
Maxsctnbr (maximum section number)	Optional	Optional	Optional

Table B-38 Reply Objects for the ENDBND Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqlcard (SQLCA reply data)	Required	Required	Required

The ENDQRYRM Reply Message

Table B-39 ENDQRYRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Required	Required	Required
Rdbnam (relational database name)	Optional	Optional	Optional
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The ENDUOWRM Reply Message

Table B-40 ENDUOWRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Required	Required	Required
Uowdsp (unit of work disposition)	Required	Required	Required
Rdbnam (relational database name)	Optional	Optional	Optional
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The EXCSAT Command

Table B-41 EXCSAT Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Extnam (external name)	Required	Required	Required
Mgrlvlls (manager level list)	Required	Required	Required
Spvnam (supervisor name)	Optional	Optional	Optional
Srvclsnm (server class name)	Required	Required	Required
Srvnam (server name)	Required	Required	Required
Srvrlslv (server release level)	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable

Table B-42 EXCSATRD Reply Object Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Extnam (external name)	Required	Required	Required
Mgrlvlls (manager level list)	Required	Required	Required
Srvclsnm (server class name)	Required	Required	Required
Srvnam (server name)	Required	Required	Required
Srvrlslv (server release level)	Optional	Optional	Optional

The EXCSQLIMM Command

Table B-43 EXCSQLIMM Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Rdbnam (name of remote database as in ACCRDB)	Optional	Optional	Optional
Pkgnamcsn (package name, consistency token and section number)	Required	Required	Required
Rdbcmtok (RDB commit allowed)	Not defined	Optional	Optional

Table B-44 Command Objects for the EXCSQLIMM Command

Command Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqlstt (SQL statement)	Required	Required	Required

Table B-45 Reply Objects for the EXCSQLIMM Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqlcard (SQLCA reply data)	Required	Required	Required

The EXCSQLSTT Command

Table B-46 EXCSQLSTT Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Rdbnam (name of remote database as in ACCRDB)	Optional	Optional	Optional
Pkgnamcsn (package name, consistency token and section number)	Required	Required	Required
Outexp (output expected)	Optional	Optional	Optional
Nbrrow (Number of insert rows)	Not defined	Optional	Optional
Prcnam (Procedure name)	Not defined	Optional	Optional
Qryblksz (Query block size)	Not defined	Not defined	Optional
Maxrslcnt (Maximum result set count)	Not defined	Not defined	Optional
Maxblkext (Maximum number of extra blocks)	Not defined	Not defined	Optional
Rslsetflg (Result set flags)	Not defined	Not defined	Optional
Rdbcmtok (RDB commit allowed)	Not defined	Optional	Optional

Table B-47 Command Objects for the EXCSQLSTT Command

Command Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqldta (SQL program variable data)	Optional	Optional	Optional

Table B-48 Reply Objects for the EXCSQLSTT Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqlcard (SQLCA reply data)	Required / Mutually Exclusive	Required / Mutually Exclusive	Required / Mutually Exclusive
Sqldtard (SQL data reply data)	Required / Mutually Exclusive / Conditional	Required / Mutually Exclusive / Conditional	Required / Mutually Exclusive / Conditional
Qrydsc (Query answer set description)	Not defined	Not defined	Optional
Qrydta (Query answer set data)	Not defined	Not defined	Optional
Sqlrslrd (SQL result set reply data)	Not defined	Not defined	Optional
Sqlcinrd (SQL result set column information reply data)	Not defined	Not defined	Optional

The MGRDEPRM Reply Message

Table B-49 MGRDEPRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Required	Required	Required
Rdbnam (relational database name)	Optional	Optional	Optional
Deperrcd (manager dependency error code)	Required	Required	Required
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The MGRLVLRM Reply Message

Table B-50 MGRLVLRM Reply Message Instance Variables

Instance Variable	**SQLAM Level 3**	**SQLAM Level 4**	**SQLAM Level 5**
Svrcod (severity code)	Required	Required	Required
Mgrlvla (manager level list)	Required	Required	Required
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The OBJNSPRM Reply Message

Table B-51 OBJNSPRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Required	Required	Required
Codpnt (code point attribute)	Required	Required	Required
Rdbnam (relational database name)	Optional	Optional	Optional
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The OPNQFLRM Reply Message

Table B-52 OPNQFLRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Required	Required	Required
Rdbnam (relational database name)	Required	Required	Required
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The OPNQRY Command

Table B-53 OPNQRY Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Rdbnam (name of remote database as in ACCRDB)	Optional	Optional	Optional
Pkgnamcsn (package name, consistency token and section number)	Required	Required	Required
Qryblksz (query block size)	Required	Required	Required
Qryblkctl (query block protocol control)	Optional	Optional	Optional
Maxblkext (Maximum number of extra blocks)	Not defined	Not defined	Optional

Table B-54 Command Objects for the OPNQRY Command

Command Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqldta (input variable data)	Optional	Optional	Optional

Table B-55 Reply Objects for the OPNQRY Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqlcard (SQLCA reply data)	Optional	Optional	Optional
Qrydsc (query answer set description)	Required	Required	Required
Qrydta (query answer set data)	Optional/ Conditional	Optional/ Conditional	Optional/ Conditional

The OPNQRYRM Reply Message

Table B-56 OPNQRYRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Required	Required	Required
Qryprctyp (query protocol type)	Required	Required	Required
Sqlcsrhld (hold cursor position)	Optional	Optional	Optional
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The PKGBNARM Reply Message

Table B-57 PKGBNARM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Required	Required	Required
Rdbnam (relational database name)	Required	Required	Required
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The PKGBPARM Reply Message

Table B-58 PKGBPARM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Required	Required	Required
Rdbnam (relational database name)	Required	Required	Required
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The PRCCNVRM Reply Message

Table B-59 PRCCNVRM Reply Message Instance Variables

Instance Variable	**SQLAM Level 3**	**SQLAM Level 4**	**SQLAM Level 5**
Svrcod (severity code)	Required	Required	Required
Prccnvcd (conversational protocol error code)	Required	Required	Required
Rdbnam (relational database name)	Optional	Optional	Optional
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The PRMNSPRM Reply Message

Table B-60 PRMNSPRM Reply Message Instance Variables

Instance Variable	**SQLAM Level 3**	**SQLAM Level 4**	**SQLAM Level 5**
Svrcod (severity code)	Required	Required	Required
Codpnt (code point attribute)	Required	Required	Required
Rdbnam (relational database name)	Optional	Optional	Optional
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The PRPSQLSTT Command

Table B-61 PRPSQLSTT Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Rdbnam (name of remote database as in ACCRDB)	Optional	Optional	Optional
Pkgnamcsn (package name, consistency token and section number)	Required	Required	Required
Rtnsqlda (specifies if SQLDA should be returned)	Optional	Optional	Optional

Table B-62 Command Objects for the PRPSQLSTT Command

Command Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqlstt (SQL Statement)	Required	Required	Required

Table B-63 Reply Objects for the PRPSQLSTT Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqldard (SQLDA reply data)	Required / Mutually Exclusive / Conditional	Required / Mutually Exclusive / Conditional	Required / Mutually Exclusive / Conditional
Sqlcard (SQLCA reply data)	Required / Mutually Exclusive / Conditional	Required / Mutually Exclusive / Conditional	Required / Mutually Exclusive / Conditional

The QRYNOPRM Reply Message

Table B-64 QRYNOPRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Required	Required	Required
Rdbnam (relational database name)	Required	Required	Required
Pkgnamcsn (package name, consistency token, and section number)	Required	Required	Required
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The QRYPOPRM Reply Message

Table B-65 QRYPOPRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Required	Required	Required
Rdbnam (relational database name)	Required	Required	Required
Pkgnamcsn (package name, consistency token, and section number)	Required	Required	Required
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The RDBACCRM Reply Message

Table B-66 RDBACCRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Required	Required	Required
Rdbnam (relational database name)	Required	Required	Required
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The RDBAFLRM Reply Message

Table B-67 RDBAFLRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Required	Required	Required
Rdbnam (relational database name)	Required	Required	Required
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The RDBATHRM Reply Message

Table B-68 RDBATHRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Required	Required	Required
Rdbnam (relational database name)	Required	Required	Required
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The RDBCMM Command

Table B-69 RDBCMM Command Instance Variable

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Rdbnam (name of remote database as in ACCRDB)	Optional	Optional	Optional

Table B-70 Reply Objects for the RDBCMM Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqlcard (SQLCA reply data)	Required	Required	Required

The RDBNACRM Reply Message

Table B-71 RDBNACRM Reply Message Instance Variables

Instance Variable	**SQLAM Level 3**	**SQLAM Level 4**	**SQLAM Level 5**
Svrcod (severity code)	Required	Required	Required
Rdbnam (relational database name)	Required	Required	Required
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The RDBNFNRM Reply Message

Table B-72 RDBNFNRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Required	Required	Required
Rdbnam (relational database name)	Required	Required	Required
Srvdgn (server diagnostic information)	Optional	Optional	Optional

The RDBRLLBCK Command

Table B-73 RDBRLLBCK Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Rdbnam (name of remote database as in ACCRDB)	Optional	Optional	Optional

Table B-74 Reply Objects for the RDBRLLBCK Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqlcard (SQLCA reply data)	Required	Required	Required

The RDBUPDRM Reply Message

Table B-75 RDBUPDRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Not defined	Required	Required
Rdbnam (relational database name)	Not defined	Required	Required
Srvdgn (server diagnostic area)	Not defined	Optional	Optional

The REBIND Command

Table B-76 REBIND Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Rdbnam (name of remote database as in ACCRDB)	Optional	Optional	Optional
Pkgnam (package name)	Required	Required	Required
Vrsnam (package version name)	Optional	Optional	Optional
Pkgisolvl (package isolation level)	Optional	Optional	Optional
Bndexpopt (bind explain option)	Optional	Optional	Optional
Pkgownid (package owner identification)	Optional	Optional	Optional
Rdbrlsopt (RDB release option)	Optional	Optional	Optional
Bndchkexs (bind existence checking)	Optional	Optional	Optional
Dftrdbcol (default RDB collection identifier)	Optional	Optional	Optional
Dgrioprl (degree of I/O Parallelism)	Not defined	Optional/ Ignorable	Optional/ Ignorable
Pkgathrul (package authorization rules)	Not defined	Not defined	Optional/ Ignorable

Table B-77 Command Objects for the REBIND Command

Command Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Bndopt (bind option)	Not defined	Not defined	Optional

Table B-78 Reply Objects for the REBIND Command

Command Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Typdefnam (data type definition name)	Optional	Optional	Optional
Typdefovr (data type definition override)	Optional	Optional	Optional
Sqlcard (SQLCA reply data)	Required	Required	Required

The RSCLMTRM Reply Message

Table B-79 RSCLMTRM Reply Message Instance Variables

Instance Variable	**SQLAM Level 3**	**SQLAM Level 4**	**SQLAM Level 5**
Svrcod (severity code)	Required	Required	Required
Rdbnam (relational database name)	Required / Conditional	Required / Conditional	Required / Conditional
Rscnam (resource name information)	Optional	Optional	Optional
Rsctyp (resource type information)	Optional	Optional	Optional
Prdid (product-specific identifier)	Optional	Optional	Optional
Rsncod (reason code information)	Optional	Optional	Optional
Srvdgn (server diagnostic area)	Optional	Optional	Optional

The RSLSETRM Reply Message

Table B-80 RSLSETRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Not defined	Not defined	Optional
Pkgsnlst (RDB Package name, consistency token, and section number list)	Not defined	Not defined	Optional
Srvdgn (Server diagnostics)	Not defined	Not defined	Optional

The SECCHK Command

Table B-81 SECCHK Command Instance Variables

Instance Variable	**SECMGR Level 5**
Secmgrnm (security manager name)	Optional / Ignorable
Secmec (Security mechanism)	Required
Password (Password)	Optional / Conditional
Newpassword (New Password)	Optional / Conditional
Usrid (Userid)	Optional / Conditional

Table B-82 Command Objects for the SECCHK Command

Command Object	**SECMGR Level 5**
Sectkn (security token)	Optional / Conditional

Table B-83 Reply Objects for the SECCHK Command

Reply Object	**SECMGR Level 5**
Sectkn (security token)	Optional / Conditional

The SECCHKRM Reply Message

Table B-84 SECCHKRM Reply Message Instance Variables

Reply Object	SECMGR Level 5
Svrcod (severity code)	Required
Secchkcd (security check code)	Required
Svcerrno (error number)	Optional / Ignorable / Conditional
Srvdgn (Server diagnostics)	Optional

The SQLERRRM Reply Message

Table B-85 SQLERRRM Reply Message Instance Variables

Instance Variable	**SQLAM Level 3**	**SQLAM Level 4**	**SQLAM Level 5**
Svrcod (severity code)	Required	Required	Required
Rdbnam (relational database name)	Optional	Optional	Optional
Srvdgn (server diagnostic area)	Optional	Optional	Optional

The SYNCCTL Command

Table B-86 SYNCCTL Command Instance Variables

Instance Variable	**SECMGR Level 5**
Synctype (sync point operation type)	Required
Rlsconv (Release Conversation)	Optional
Uowid (Unit of Work Identifier)	Optional
Forget (Forget Unit of Work)	Optional

Table B-87 Command Objects for SYNCCTL

Command Object	**SECMGR Level 5**
Synclog (Sync point log)	Optional

The SYNCCRD Reply Object

Table B-88 SYNCCRD Reply Object Instance Variables

Instance Variable	**SECMGR Level 5**
Synctype (Sync point operation type)	Required

The SYNCLOG Reply Object

Table B-89 SYNCLOG Reply Object Instance Variables

Command Object	**SECMGR Level 5**
Rdbnam (Relational Database Name)	Required
Logname (Log Name)	Required
Logtstmp (Log Timestamp)	Required
Cnntkn (Connection Token)	Required
Snaaddr (SNA Resync Address)	Optional — mutually exclusive with Ipaddr)
Ipaddr (TCP/IP Resync Address)	Optional — mutually exclusive with Snaaddr)
Tcphost (TCP/IP Domain Qualified Host Name)	Optional — mutually exclusive with Snaaddr)

The SYNCRSY Command

Table B-90 SYNCRSY Command Instance Variables

Instance Variable	SECMGR Level 5
Rsynctyp (Resync Type)	Required
Uowid (Unit of Work Identifier)	Optional
Uowstate (Unit of Work State)	Optional

Table B-91 Command Objects for SYNCRSY

Command Object	SECMGR Level 5
Synclog (Sync point log)	Optional

The SYNCRRD Reply Object

Table B-92 SYNCRRD Reply Object Instance Variables

Instance Variable	SECMGR Level 5
Rsynctyp (Resync Type)	Required
Uowid (Unit of Work Identifier)	Optional
Uowstate (Unit of Work State)	Optional

Table B-93 Reply Objects for SYNCRRD

Reply Object	SECMGR Level 5
Synclog (Sync point log)	Optional

The SYNTAXRM Reply Message

Table B-94 SYNTAXRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Required	Required	Required
Synerrcd (syntax error code)	Required	Required	Required
Codpnt (code point attribute) Rdbnam (relational database name)	Optional Optional	Optional Optional	Optional Optional
Srvdgn (server diagnostic area)	Optional	Optional	Optional

The TRGNSPRM Reply Message

Table B-95 TRGNSPRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5
Svrcod (severity code)	Required	Required	Required
Rdbnam (relational database name)	Optional	Optional	Optional
Srvdgn (server diagnostic area)	Optional	Optional	Optional

The VALNSPRM Reply Message

Table B-96 VALNSPRM Reply Message Instance Variables

Instance Variable	**SQLAM Level 3**	**SQLAM Level 4**	**SQLAM Level 5**
Svrcod (severity code)	Required	Required	Required
Codpnt (code point attribute)	Required	Required	Required
Rdbnam (relational database name)	Optional	Optional	Optional
Srvdgn (server diagnostic area)	Optional	Optional	Optional

Glossary

This glossary defines terms as they are used for DRDA. If a term is not included here, see the other references listed in **Referenced Documents** on page xxiv about that topic.

alert
An error message sent to the system services control point (SSCP) at the host system.

API
Application Programming Interface.

Application Programming Interface (API)
The interface that application programs use to request services from some program such as a DBMS.

application requester (AR)
The source of a request to a remote relational database management system (DBMS).

application server (AS)
The target of a request from an application requester. The DBMS at the application server site provides the data.

application support protocol
The protocol that connects application requesters and application servers.

AR
Application Requester.

AS
Application Server.

bind
In DRDA, the process by which the SQL statements in an application program are made known to a DBMS over Application Support Protocol flows. During a bind, output from a precompiler or preprocessor is converted to a control structure called a package.

CCSID
Coded Character Set Identifier.

CDRA
Character Data Representation Architecture. The architecture that defines CCSID values to identify the codes (code points) used to represent characters, and the (character data) conversion of these codes, as needed, to preserve the characters and their meanings.

Coded Character Set Identifier (CCSID)
A 16-bit number identifying a specific set of encoding scheme identifiers, character set identifiers, code page identifiers, and other relevant information that uniquely identifies the coded graphic character representation used.

commit on return
An attribute of a stored procedure definition indicating that the transaction is to be committed immediately upon successful (that is, no negative SQLCODE) return from the stored procedure.

connectivity
A technology that enables different systems to communicate with each other.

conversation
A logical connection between two programs over an LU type 6.2 session that allows them to communicate with each other while processing a transaction.

data integrity

1. Within the scope of a unit of work, either all changes to the database management systems are completed or none of them are. The set of change operations are considered an integral set.
2. The condition that exists as long as accidental or intentional destruction, alteration, or loss of data does not occur.

database-directed distributed unit of work
A variant of distributed unit of work in which a user or application directs SQL statements to a targeted DBMS, which then directs the SQL statement, if needed, to another DBMS for execution at the DBMS. As in distributed unit of work, the user or application can, within a single unit of work, read and update data on multiple DBMSs. Each SQL statement may access only one DBMS.

Database Management System (DBMS)
An integrated set of computer programs that collectively provide all of the capabilities required for centralized management, organization, and control of access to a database that is shared by many users.

database server (DS)
The target of a request received from an application server.

database support protocol
The protocol used to connect application servers and database servers.

DBCS
Double-byte character set.

DBMS
Database management system.

DCE
Distributed Computing Environment

DDM
Distributed Data Management Architecture. The architecture that allows an application program to work on data that resides in a remote system. The data may be in files or in relational databases. DRDA is built on the DDM architecture.

Distributed Computing Environment (DCE)
The name of the distributed environment developed by the Open Software Foundation. DCE is composed of common services required to provide an open distributed computing environment.

distributed request
An extension of the distributed unit of work method of accessing distributed relational data in which each SQL statement may access data located at several different systems. This method supports join and union operations that cross system boundaries and inserts of data selected from other sites.

distributed unit of work
A method of accessing distributed relational data in which a user or application can, within a single unit of work, read and update data on multiple DBMSs. The user or application

directs each SQL statement to a particular DBMS for execution at that DBMS. Each SQL statement may access only one DBMS.

double-byte character set (DBCS)
A character set, such as a set of Japanese ideographs, that requires two-byte code points to identify the characters.

DRDA
Distributed Relational Database Architecture. A connection protocol for distributed relational database processing. DRDA comprises protocols for communication between an application and a remote database, and communications between databases. DRDA provides the connections for remote and distributed processing.

DRDA Connection
A connection between an application requester and application server for the purposes of performing DRDA requests. A DRDA connection generically includes any other connections that are required to allow an application requester and application server to communicate (for example, network connection, SQL connection, and so on). DRDA is sometimes qualified with a numeric value (that is, 1, 2, and so on) to indicate the connection supports that level of DRDA.

DS
Database server.

dynamic SQL
SQL statements that are prepared and executed within a program while the program is executing. In dynamic SQL, the SQL source is contained in host language variables rather than being coded in the application program. The SQL statement might change several times during the program's execution.

execution
The process of carrying out an instruction or instructions of a computer program by a computer.

execution thread
A process or task that provides for the execution of a sequence of operations. One operation occurs at a time. Operations are single threaded. Commonly, resources (such as locks) are associated with execution threads, and the thread becomes the anchor point for managing such resources.

Extended Privilege Attribute Certificate (EPAC)
A DCE construct that contains Extended Registry Attributes in addition to the principal's identity and group memberships.

Extended Registry Attribute (ERA)
A user-defined attribute in the DCE Security Registry. Each ERA has a schema entry that is the data dictionary entry defining the attribute type. Instances of the attribute containing values can be attached to principal, group, organization, or policy nodes in the DCE Security Registry database.

flow
The passing of a message from one process to another. The passing of messages of a particular type between processes. For example, DRDA flows are those that consist only of messages described by DRDA as part of the DRDA protocols.

FD:OCA
Formatted Data Object Content Architecture. An architected collection of constructs used to interchange formatted data.

GSS-API

Generic Security Services-Application Programming Interface. A programming interface for accessing generic security services. GSS-API is available in DCE for utilizing DCE security outside of RPC.

host variable

In an application program, a program variable referenced by SQL statements.

instantiate

To create an instance of something.

LID

Local identifier

like

Two or more similar or identical operating environments. For example, like distribution is distribution between two OS/2 database managers with compatible server attribute levels.

local identifier (LID)

An identifier or short label that is mapped by the environment to a named resource.

logical unit (LU)

A port through which an end user accesses the SNA network in order to communicate with another end user and through which the end user accesses the functions provided by system services control points (SSCP).

logical unit of work (LUW)

The work that occurs between the start of a transaction and commit or rollback and between commit and rollback actions after that. It defines the set of operations that must be considered part of an integral set. See data integrity.

logical unit-of-work identifier (LUWID)

A name—consisting of a fully-qualified LU network name, an LUW instance number, and an LUW sequence number—that uniquely identifies a logical unit of work within a network.

Logical Unit type 6.2 (LU 6.2)

The SNA logical unit type that supports general communication between programs in a distributed processing environment.

LU

Logical unit.

LU 6.2

Logical Unit type 6.2.

LUW

Logical unit of work.

LUWID

Logical unit of work identifier.

MBCS

Mixed-byte character set.

mixed-byte character set (MBCS)

A character set containing a mixture of characters from single byte and double byte character sets.

MSA

SNA Management Services Architecture. The architecture that provides services to assist in the management of SNA networks.

Mutual Authentication
The name of the authentication process where the server authenticates the client and the client authenticates the server.

network connection
A logical connection between two endpoints in a network. A network connection allows the two endpoints to communicate.

package
The control structure produced when the SQL statements in an application program are bound to a relational DBMS. The DBMS uses the control structure to process SQL statements encountered during statement execution.

plan
A form of package where several programs' SQL statements are collected together during bind to create a plan. DRDA does not support the concept of plan.

port
A term used in TCP/IP that specifies the portion of a socket that identifies the logical input or output channel associated with a process.

principal
An entity whose identity can be authenticated. In terms of DRDA and DCE, this would be the end user initiating a database request.

program preparation process
That process, usually involving programmers, whereby a program is written, possibly precompiled, compiled, possibly link-edited, and bound. Thus, the program is made available for execution. This process and the tools available to assist in this process vary greatly among the various systems that may support DRDA.

protected conversation
A protected conversation is an LU 6.2 conversation that supports two-phase commit protocols for resource recovery.

protected network connection
A network connection that is supported by protocols that allow for coordinated resource recovery (for example, two-phase commit protocols).

protected resource
A resource that is updated in a synchronized manner during resource recovery processing.

protocol
The rules governing the functions of a communication system that must be followed if communication is to be achieved.

RDB
Relational database. All the data that can be accessed via RDB_NAME. For example, a catalog and all the data described therein, or for OS/400, all collections with their associated catalogs as well as all other database libraries on a particular system.

RDB_NAME
The DRDA globally unique name for an RDB.

relational data
Data stored in a relational database management system.

remote unit of work
The form of SQL distributed processing where the application is on a system different from the RDB. A single application server services all remote unit of work requests within a single unit of work.

Replay
A security attack in which a perpetrator observes valid authentication information that is passed between two partners, and then uses that information to gain access to one of the partners by sending the exact same information.

resource recovery
The process that allows logical units of work to set new synchronization points, or to allow a unit of work to roll back to the most recently established synchronization point. In LU 6.2 terms, this is sometimes known as synchronization point processing.

robust
A characteristic of a network protocol that provides functions required by DRDA. For example, instant notification to both parties of a connection when failure occurs to either party or the connection between them.

SBCS
Single-byte character set.

security context information
A string of bytes received from a GSS--API call (*gss_init_sec_context*( ) and *gss_accept_sec_context*( )) to be used to set up a security context between an application requester and application server. Setting up a security context includes verifying the partner. This is also known as identification and authentication.

semantics
The part of a construct's description that describes the function of the construct.

single-byte character set (SBCS)
A character set that requires one-byte code points to identify the characters.

SNA
Systems Network Architecture.

socket
A term used in TCP/IP that specifies an address which specifically includes a port identifier; that is, the concatenation of an Internet Address with a TCP port.

SQL
Structured Query Language. A standardized language for defining and manipulating data in a relational database.

SQL Connection
An SQL connection is a logical connection between an SQL application program and a DBMS where the SQL application issues SQL calls to perform database functions.

SSCP
System services control point.

synchronization point (sync point)
The beginning or end of a unit of work. It is used as a reference point to which resources can be restored if a failure occurs during the unit of work.

synchronization point manager
The component of an operating environment that coordinates commit and rollback

operations on protected resources.

system services control point (SSCP)
A focal point within an SNA network for managing the configuration, coordinating network operator and problem determination requests, and providing directory services and other session services for end users of a network.

Systems Network Architecture (SNA)
The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through and controlling the configuration and operation of networks.

target program name (TPN)
The name by which a program participating in a network connection is known. Normally, the initiator of a network connection will identify the name of the program it wishes to connect to (see transaction program name).

TCP/IP
Transmission Control Protocol/Internet Protocol. An Internet standard transport protocol that provides reliable, full duplex, stream service.

TP
Transaction program.

TPN
Target program name. See target program name, transaction program name, and well known port.

transaction
See Logical Unit of Work

transaction program (TP)
A program that processes transactions in an SNA network. There are two kinds of transaction programs: application transaction programs and service transaction programs.

transaction program name
The name by which each program participating in an LU 6.2 conversation is known. Normally, the initiator of a conversation will identify the name of the program it wishes to connect to at the other LU. When used in conjunction with an LU name, it identifies a specific transaction program in the network.

triplet
An FD:OCA triplet consists of three parts:

1. a length byte
2. a type byte
3. one or more parameter-value bytes

Triplets are referred to by their type, such as Row LayOut triplet (RLO). Triplets may refer to other triplets using LIDs.

two-phase commit protocols
The protocols used by a sync point manager to accomplish a commit operation.

unit of work
A sequence of SQL commands that the database manager treats as a single entity. The database manager ensures the consistency of data by verifying that either all the data changes made during a unit of work are performed or none of them are performed.

Universal Unique Identifier (UUID)
A DCE term that identifies a unique identifier of an end user. A DCE realm has a unique identifier in the set of realms. Userids within a realm have unique identifiers. If the realm UUID is included with the end-user UUID, the resultant UUID is universally unique.

unlike
Two or more different operating environments. For example, unlike distribution is distribution between DB2 for VM and DB2 for MVS.

unprotected conversation
An unprotected conversation is an LU 6.2 conversation that does not support two-phase commit protocols for coordinated resource recovery.

unprotected network connection
A network connection that is not supported by protocols that allow for coordinated resource recovery (for example, two-phase commit protocols).

UUID
Universal Unique Identifier.

well-known port
A port that is registered with the Internet as providing a specified type of support (for example, DRDA application server).

Index

crrtkn **60**, 415
extnam **58**
qryblksz **80**, 83
srvclsnm (server class name).......... **57**
srvdgn **53**
sttstrdel **60**
svrcod **53**
typdefovr **61**
ABEND **116**, 353, 408, 415
ABNUOWRM **110**, 135, 294, 318, 331
access path **273**
access relational database **44**
accounting **342**
accounting information **26**, 387, 392, 424
ACCRDB **44**, 52, 54, 59, 145, 162
.... 200, 256, 395, 400, 425
ACCRDBRM **60-**61, 81, 152, 162, 282, 290, 331, 400
ACCSEC **62**, 397, 425
ACCSECRD **62**, 397, 425
actions **359**
additional SV **360**
agent **41**, 49-50, 56
AGNPRM alert **358**
AGNPRMRM **303**, 331
alert 507
alert descriptions **359**
alert example **375**
alert generation **354**
alert implementation basics **354**
alert model mapping **355**
alert models **357**
alert structures **354**
alert to reply message mapping **355**
alert types **359**
alerts **354**
alerts and supporting data **415**
alerts at application requester **357**
alias **278**
ALLOCATE **270**, 390, 392
alphabetic extender **271**
API 507
API (Application Programming Interface).......... **20**
APPC **387**
APPCMNI **392**
APPCMNT **408**
application program version management **274**
application programming interface (API).......... **20**
Application Programming Interface (API).......... 507
application requester 40
application requester (AR).......... **22**, 25, 40, 54, 507
application server 40
application server (AS) **22**, 25, 54, 507
application services **423**
application support protocol **22**, 25, 507
APPSRCCD **401**
APPSRCCR **401**
AR 507
array description **163-**165
array descriptor **166**
AS 507
assigning LIDs to 0 triplets **252**
assigning override **252**
asynchronous wait **395**
ATTACH **394**
authenticated conversation **25**
authentication **345**, 387, 392, 424
authorization **387**
BACKOUT **391**, 413
base and option sets **138**
base row data 313
base set functions **390**
base set of LU 6.2 .... **389**
basic conversation **415**
basic conversation verb **391**
basic FD:OCA object contained in DDM **138**
BF rule **310**
BGNBND **44**, 67, 69, 145, 402, 426
BGNBNDRM **69**, 331
bind **29**, 31, 273, 507
bind flow **67**, 69, 71, 402, 426
bind flows **402**, 426
bind option values **4**
bind options **8**, 69
BLKERR alert **362**
block chaining rule (CH rule) **314**
block format rule (BF rule) **310**
block size rule (BS rule) **313**
blocking **310**
blocking protocol error **357**, 362
blocking rule **310**
BNDOPT **69**, 75, 298
BNDSQLSTT **44**, 67, 69-70, 118, 145, 402, 426

BNDSTTASM....**297**
BS rule....**313**
building statement level SQLCAs....**437**, 440
CA rule....**281**-282
causes....**359**
CCSID....**11**, 50, 52, 59, 61, 162
....199, 244, 253, 282, 287, 290, 292, 307
....334, 507
CCSID manager....**46**, 49
CCSIDDBC....**282**
CCSIDMBC....**282**
CCSIDSBC....**282**
CD rule....**283**
CDRA....**19**, 52, 199, 507
CF rule....**283**
CH rule....**314**
chaining rule (CH rule)....**314**
chaining violation....**357**
character data representation architecture....**19**
....52, 199
character set restriction....**334**
CHNVIO alert....**363**
CLSQRY....**31**, 44, 78, 82, 87, 145
CMDATHRM....**331**
CMDCHK alert....**364**
CMDCHKRM....**303**, 331
CMDNSPRM....**295**, 331
CMDVLT alert....**365**
CMDVLTRM....**132**-133, 285, 303, 331
CMMRQSRM....**122**, 127-129, 132-133, 285, 331
CMNAPPC....**40**, 49, 385
CMNMGR....**341**
CMNSYNCPT....**41**, 49, 385
CMNTCPIP manager....**421**
CMNTCPIP manager)....**41**
CNTQRY....**31**, 44, 78, 81-82, 143, 145, 312
code point....**49**, 143
coded character set identifier....**11**, 50
....52, 59, 61, 162, 199, 244, 253, 282
....287, 290, 292, 307, 334
Coded Character Set Identifier (CCSID)....507
coded character set identifier manager....**46**
codepoint....**52**
CODPNT....**52**
CODPNTDR....**52**
coexistence....**120**-121, 124
COLLECTION....**271**, 273
command and reply flow....**54**
command data object....**44**
command execution example....**259**
command to descriptor relationship....**145**
commit....**32**
COMMIT....**99**, 116, 121-122, 125, 127-128, 131
commit....**134**
commit and rollback....**118**
commit and rollback processing rule....**284**
commit and rollback rule....**284**-285
commit and rollback scenario....**124**
commit flow....**117**-118
commit flows....**429**
  on SYNC_LEVEL(NONE) conversation....410
  on SYNC_LEVEL(SYNCPT) conversation....411
commit on return....507
commit unit of work....**116**
commit unit of work DDM flow....**117**
commit/rollback processing....**416**, 434
commitment of work....**116**
communication area
  group description....190
  row description....175
communication area exceptions
  group description....191
communication connection....**299**
communications manager....**40**, 49
....51, 55-56, 59, 341, 385, 421
CONNECT statement....**19**, 25-26
connection allocation....**415**
connection allocation rule (CA rule)....**281**-282
connection allocation rules....**434**
connection failure (CF rule)....**283**
connection usage rule (CU rule)....**287**
connectivity....**20**, 38, 507
consistency token....**274**
continue preceding triplet....**138**, 149, 157
control-operator verb....**388**
conversation....**392**, 508
conversation allocation rule....**282**
conversation failures....**413**
conversation flow....**282**, 316
conversation level....**390**
conversation protocol error....**368**
conversation rule....**281**-282, 287
conversation verb....**388**
conversation verb category....**390**
conversation-level security....**390**, 392
conversational communications manager....**40**
....50, 385
correlation....**352**
correlation displays....**353**
correlation of diagnostic information....**414**
correlation token....**351**, 353, 415, 434
correlation, alerts and supporting data....**415**

correlation, focal point messages........................**353**
CPT **138**, 149, 157
CR rule...**284**
creating a package **67**
CU rule...**287**
cursor error condition...**369**
data area group description.........................**192**, 194
data area row description.....................................**176**
data array description...**153**
data collection **415**
data definition and exchange...............................**137**
data descriptor **143**
data integrity 508
data representation transformation....................**291**
data representation transformation rule...**290**, 292
data staging area **8**
data stream structure error...................................**357**
data stream syntax error **374**
data type conversion rule (DC rule)...................**288**
data types **3**
data value group description...............................**159**
Database Management System (DBMS)............508
database server..40
database server (DS) 508
database support protocol....................................508
database-directed access **3**
database-directed distributed unit of work......508
date description **216**
DBCS 508
DBCS (GRAPHIC) **227**-229
DBMS 508
DC rule...**288**
DCE................................**25**, 61-64, 299, 417, 435, 508
DCE security **343**, 417
DDM.......................................**26**, 31, 38-40, 44, 49-50
..............................52, 54, 69, 79, 118, 275, 385, 402
....421, 426, 508
DDM bind flow **426**
DDM command objects in DRDA.........................**15**
DDM concepts..**13**
DDM concepts for DRDA implementation.........**14**
DDM reader guide..**13**
DDM servers..40
deadlocks...**116**
DEALLOCATE..............**353**, 357, 390, 407, 415, 417
deallocation type...................................**353**, 408, 415
decimal description **207**-208
default triplet..160
degrees of distribution...**22**
DESCRIBE **109**
describe input **2**
describe table **112**
descriptor classes...**149**
descriptor definitions..**151**
descriptor object **138**
dgrioprl **68**
diagnostic data collection **415**
diagnostic information
  collection and correlation **353**
diagnostic support..**52**
dictionary manager..**42**, 49
directory manager **42**
disconnect **303**
display..**414**
Distributed Computing Environment (DCE)...508
Distributed Data Management............................**385**
distributed request...**22**, 508
distributed unit of work **9**
Distributed Unit of Work **20**
distributed unit of work...............................**430**, 508
double-byte character set (DBCS) 509
DRDA **281**, 509
DRDA Connection 509
DRDA managers...**40**
DRDA rules...**281**
drop package **72**
DRPPKG..**44**, 72, 145
DS..509
DSCERR alert **366**
DSCINVRM...**254**, 303, 331
DSCPVL..**145**
DSCRDBTBL **44**, 112-113, 145
DSCSQLSTT **44**, 110-111, 145
DT rule **290**
DTAMCHRM **177**, 179, 181, 183
..............185-186, 188, 191-192, 194, 254, 303, 331
dynamic commit and rollback............................**116**
dynamic execution **114**
dynamic rollback...................................**128**, 130, 132
dynamic SQL.....................**60**, 108-109, 114-115, 509
dynamic SQL scenario...........................**127**-128, 131
early array descriptor **162**
early data unit descriptors...................................**258**
early descriptor **143**
early descriptors **162**
early environmental description **256**
early environmental descriptor...........**197**, 199-200
early environmental descriptors **197**
early group descriptors **177**
early row descriptors...**168**
eight-byte description..**205**
eight-byte integer...**211**

end user **417**
end user name **270**
end-user name.....**270**
end-user name rule (EUN rule) **307**
ENDBND **44**, 67, 71, 145, 275, 402, 426
ENDQRYRM **78**, 81, 84-85, 87-88, 105
.....107, 263, 331, 406, 427
ENDUOWRM **117**, 119, 124, 126, 133
.....284, 331, 410, 416, 429
enhanced bind options **8**
enhanced security.....**6**
enhanced sync point manager **7**
enterprise code.....**271**
environmental description.....**141**, 143
environmental description objects.....**256**
environmental descriptor **149**
error checking.....**254**
error condition to alert model mapping.....**355**
error reporting.....**254**
EUN rule.....**307**
exchange server attribute **52**, 54
.....58, 62, 143, 287, 395, 400, 425
EXCSAT **52**, 54, 58, 62, 143, 287, 395, 400, 425
EXCSAT (exchange server attribute).....**162**
EXCSATRD **52**, 54, 58, 62, 162
EXCSQLIMM.....**44**, 114-115, 121, 145, 260, 284
EXCSQLSTT.....**44**, 97, 100, 108, 118-119
.....143, 145, 284, 395, 400, 425
execute SQL statement **97**
EXECUTE_IMMEDIATE.....**260**
execution 509
execution flow **405**-406, 427
execution thread **52**, 509
Extended Privilege Attribute Certificate (EPAC)509
Extended Registry Attribute (ERA).....509
externalized row data 313
failure causes **359**
failure notification **387**
FD:OCA **39**, 137, 143, 245
.....250, 252, 254, 256, 310, 509
FDODSC.....**150**
FDODTA.....**138**
fixed byte description **219**
fixed character mixed description.....**230**
fixed character SBCS description **224**
fixed decimal description.....**207**
fixed-character description.....**227**
fixed-row protocol.....**31**, 78
FIXROWPRC **31**, 84, 316, 400, 426
float **204**-206
flow **25**, 29, 31-32, 34-35, 54, 67, 114-115, 509
focal point messages **351**, 354, 434
Formatted Data Object Content Architecture (FD:OCA)**19**
four-byte description.....**201**, 206
GDA.....**141**, 149, 157, 199-200, 252
GDA / CPT errors **255**
general errors.....**254**
generating alerts **354**
generic focal point messages.....**354**
GENERR alert.....**367**
GET_ATTRIBUTES.....**390**, 394-395
GET_TP_PROPERTIES **390**, 394-395
group data array **141**, 149, 157, 199-200, 252
GSS-API.....**343**, 510
gss_accept_security_context
function 344
gss_accept_sec_context
function 512
gss_init_security_context
function 344
gss_init_sec_context
function 343, 512
handling conversation failures **413**
hexadecimal descriptor **154**
host variable 510
I / O parallelism **11**
ID number alert.....**359**
immediate SQL statement execution **114**
initialization flow.....**25**, 395, 397, 400
initialization flows.....**424**
initializing a conversation **392**
initializing a TCP / IP connection.....**424**
INSERT command.....**264**
install causes.....**359**
instantiate.....510
integer.....**201**-203
INTRDBRQS.....**44**, 145
IR rule.....**294**
large object bytes **239**
large object bytes locator **212**
large object character
DBCS (GRAPHIC).....**241**
DBCS locator **214**
locator **213**
SBCS **240**
large object character mixed **242**
late array descriptors **152**
late data unit descriptors **259**
late descriptor **143**, 152
late environmental descriptor **199**, 243
late group descriptors.....**157**

late row descriptors **155**
level 1 ....**8**
level 2 ....**2**, 8
level 3 ....**5**
level 4 ....**2**
LID **149**, 198, 252-253, 510
LID (local identifier) **141**, 157
LID example **252**
like 510
limited block protocol **31**, 85
    LOB data....................**90**
LMTBLKPRC....................**31**, 85, 89, 313, 400, 426
LOB data....................76
local identifier....................**141**, 149, 198, 252-253
local identifier (LID)....................510
logical flow **400**, 426
logical unit (LU)....................510
logical unit of work....................**414**
logical unit of work (LUW) 510
logical unit of work identifier **414**
Logical Unit type 6.2 (LU 6.2) 510
logical unit-of-work identifier**26**, 392, 414-415, 417
logical unit-of-work identifier (LUWID)....................510
LOG_DATA....................**408**
long variable bytes description **221**
long variable character description **229**
long variable character mixed description....................**232**
long variable character SBCS description **226**
LU 510
LU 6.2....................**407**-408, 413, 416, 510
LU 6.2 (Logical Unit 6.2) **353**, 385, 388, 414, 417
LU 6.2 base and option sets....................**390**
LU 6.2 flow **400**, 402, 406
LU 6.2 initialization flow....................**395**
LU 6.2 initialization processing....................**392**
LU-LU verification **392**
LUNAME **392**
LUW 510
LUWID....................**26**, 392, 414-415, 417, 510
LUW_Identifier **391**
LUW_IDENTIFIER....................**394**
LU_NAME **392**
major subfield construction....................**375**
major subvector construction **375**
major vector construction....................**375**
managing conversation....................**413**
mapping of RMs to SQLSTATEs (CD rule)....................**283**
mapping reply messages **355**
materialization rules....................**253**
MAXBLKEXT **100**
MAXRSLCNT....................**100**
MAXSCTNBR **275**, 300
MBCS....................510
MDD **138**, 141, 149, 153, 197, 243, 247, 250, 253
MDD errors....................**254**
message models....................**354**
meta data definition....................**138**, 141
    ....149, 153, 197, 243, 247, 250, 253
meta data description....................**153**
meta data summary **245**
MGRDEPRM....................**331**, 413
MGRLVLRM....................**331**
mixed-byte character set (MBCS)....................510
mixed-byte datalink....................**238**
model mapping....................**355**
MSA....................**19**, 510
multi-RDB scenario....................**132**, 134
multi-relational database update **120**, 131, 136
multi-row fetch **10**, 316, 437, 440
multi-row insert....................**10**, 264
Multilingual Latin-1....................**243**
Mutual Authentication....................511
name syntax....................**271**
name tables and views **271**
naming conventions....................**269**, 279, 419
network connection....................**25**, 124, 428-429, 511
network connections....................**120**
network management **351**, 414
non-standard query blocks....................310
null-terminated bytes description....................**222**
null-terminated mixed description....................**233**
null-terminated SBCS description **223**
nullable **191**
number of elements group description **187**
number of elements row description **173**
numeric character description....................**209**
OBJDSS **49**, 51, 310, 315, 425
object data stream structure....**49**, 51, 310, 315, 425
object name group description....................**186**
object name row description....................**172**, 318
object name rule (ON rule)....................**307**
object-oriented extensions 4
OBJECTID **271**
OBJNSPRM....................**295**, 331
OC rule....................**295**
ON rule....................**307**
one-byte description **203**
OPNQFLRM....................**331**
OPNQRY..........**31**, 44, 78-79, 88, 143, 145, 261, 316
OPNQRYRM.................**78**, 81, 85, 87, 331, 406, 427
option set functions....................**390**
option set of LU 6.2 ....**389**

optionality (OC rule) **295**
optionality rules..**295**
override triplet...160, 252
overriding descriptor....................................**252**, 254
overriding descriptors **250**
overriding everything...**250**
overriding output formats...................................**160**
overriding user data..**252**
package **29**, 31, 67, 273, 511
package consistency token **274**
PACKAGEID **273**
Pascal L string bytes..**234**
Pascal L string description...................................**235**
Pascal L string mixed..**236**
passing USER_ID...**270**
passing warning to application requester rule.**306**
PASSWORD..**393**
PB rule **296**
pkgathrul...**69**, 75
PKGBNARM...**331**
PKGBPARM **69**, 331
plan 511
port 511
post-commit processing..**134**
POST_ON_RECEIPT **393**-394, 401
PRCCNV alert **368**
PRCCNVRM **303**, 331
PREPARE_TO_RECEIVE **391**, 393, 401
principal..511
privileges area group description **188**
privileges area repeating group
  row description **174**
PRMNSPRM..**295**, 331
probable causes..**359**
problem determination **351**, 414-415, 433-434
problem determination and isolation **353**
process model...**40**
process model flow...**49**-50
product-unique extensions.....................................**52**
program binding rule (PB rule) **296**
Program name **279**
program preparation process **275**, 511
program to program communication **387**
protected conversation..511
protected network connection.............................511
protected resource..511
PROTECTED_LUW_IDENTIFIER **391**, 394
protocol **31**, 511
PRPSQLSTT **44**, 108-109, 145, 284
PWDENC 3
PWDSBS..3
QP rule **316**
qryblkctl..**79**
qryblksz **79**
QRYDSC **86**, 88, 107, 143, 316
QRYDTA...**82**, 138, 143, 311
QRYERR alert **369**
QRYNOPRM..**87**, 303, 331
QRYPOPRM...**81**, 87, 303
QT rule **318**
query block chaining rule (CH rule)...................**314**
query block format rule (BF rule) **310**, 313
query block size rule (BS rule) **313**
query blocks..**310**
query data **138**
query data transfer protocol rule (QP rule) **316**
query flow **67**, 78, 80
query process..**76**
query termination rule **282**
query termination, interruption, continuation.**318**
RDB **45**, 511
RDB initiated rollback rule **294**
RDB initiated rollback scenario **135**
RDBACCRM **303**, 331
RDBAFLRM..**331**
RDBATHRM **331**
RDBCMM.......................................**44**, 117-118, 145
RDBERR alert **370**
RDBMS...**19**
RDBNACRM...**303**, 331
RDBNFNRM..**331**
RDBRLLBCK....................................**44**, 145, 411, 430
RDBUPDRM **121**, 305, 331
RDB_NAME...**271**, 273, 511
RDB_NAME rule (RN rule)..................................**307**
READ socket call **427**
REBIND **44**, 74-75, 145
RECEIVE operations.....................................**393**, 401
RECEIVE_AND_WAIT.........**390**, 395, 405-406, 408
referencing overrides **252**
referencing rule **252**
relational data...511
relational database access error..........................**370**
relational database commit unit of work **117**
relational database manager................**45**, 50, 56, 59
relational database name rule (RN rule) **307**
relational database names **271**
relational database names rules **435**
relational database-initiated rollback rule **294**
Remote Unit of Work **21**
remote unit of work.....................................**429**, 512
Replay 512

reply data object **44**, 50
reply data stream structure **51**, 263
.....................................310, 318, 400, 406, 425, 427
reply message **60**, 117, 406
reply objects and messages **16**
request correlation identifier...........................**49**, 51
required base set functions...................................**390**
required option set function.................................**390**
resource limit reached..................................**357**, 371
resource recovery......................................**32**, 44, 124
.....................................132, 134, 281, 284, 413, 512
resource recovery interface..........................**116**, 131
result set column information
row description..**169**
result set column information group **179**
result set group description **177**
result set locator...**210**
result set row description **168**
resynchronization manager....................................**43**
RLO **137**-138, 141, 149-150, 152, 155, 157, 252
RLO errors..**255**
RN rule...**307**
robust 512
ROLLBACK...**89**, 116
rollback **117**
ROLLBACK...**121**-122
rollback flow **430**
rollback flows
on SYNC_LEVEL(NONE) conversation **411**
on SYNC_LEVEL(SYNCPT) conversation ...**412**
rollback unit of work **117**
row identifier..**215**
row layout**137**-138, 141, 149-150, 152, 155, 157, 252
RPYDSS **51**, 263, 310, 318, 400, 406, 425, 427
RQSDSS **49**
RSCLMT alert...**371**
RSCLMTRM **331**
RSLSETFLG **100**
RSLSETRM **100**, 331
RSYNCMGR (resynchronization manager)........**43**
rule CA5..**281**
rule usage **415**-416
rule usage of relational database names............**418**
rules for CLOSE **329**
rules for CLSQRY **320**
rules for CNTQRY..**320**
rules for EXCSQLSTT **320**
rules for FETCH **325**
rules for OPNQRY..**320**
rules usage for TCP/IP environments **434**
rules usage fpr SNA..**415**
SBCS **223**-226, 235, 307, 333, 512
SBCS datalink...**237**
scrollable cursors **10**
SDA..........**137**-138, 141, 143, 149, 197, 200, 243, 252
SE rule **299**
SECCHK **62**, 397, 425
SECCHKRM **62**, 303, 331, 397, 425
SECMGR **42**, 50
section number...**275**
section number assignment rule (SN rule) **300**
security **6**, 25, 61, 270, 390, 416
security context information.......................**343**, 512
security flow **64**
security manager **41**-42, 50, 56
security mechanism
userid and encrypted password......................**349**
userid and password **345**
userid and password substitute **348**
userid, password, new password **346**
userid-only...**347**
security mechanisms..**3**
security rule (SE rule) **299**
security violation error..**373**
SECURITY_USER_ID **394**
SECVIOL alert..**373**
semantics..512
SEND_DATA..........................**390**, 393, 395, 401, 405
SEND_ERROR **390**
server list **7**
serviceability rule (SV rule)................................**303**
serviceability rules **417**, 435
SET statement (ST rule)..**302**
SET_SYNCPT_OPTIONS **391**
severity codes...**53**, 355
simple data array...................................**137**-138, 141
.....................................143, 149, 197, 200, 243, 252
single relational database update **120**
single-byte character set.......**223**-226, 235, 307, 333
single-byte character set (SBCS)..........................512
sixteen-byte description.......................................**204**
SN rule **300**
SNA..**25**-26, 35, 385, 512
SNA environment usage.......................................**414**
SNA Management Services Architecture **19**
SNA sync point communications manager **41**
socket 512
socket calls **427**
sockets interface...**423**
SP rule **301**
special register...**302**
SQL..**19**, 25-26, 31, 61, 69

...............114-116, 159, 168-177, 179, 181, 185-187
.......................190-192, 194, 196, 274, 307, 405-406
....427, 512
SQL application manager **56**
SQL Connection..512
SQL EXECUTE IMMEDIATE................................**114**
SQL EXECUTE_IMMEDIATE **275**, 284
SQL object name rule (ON rule)..........................**307**
SQL object names (ON rules)...............................**307**
SQL statement execution flow **405**, 427
SQLAM (SQL application manager) **43**, 45
....48, 51, 162
SQLCA......................................**53**, 153, 157, 166, 175
....260, 263, 283, 437, 440
SQLCADTA **153**, 157
SQLCAGRP..**157**, 175, 190
SQLCARD **67**, 71, 73, 75, 97
................................99-100, 108, 114, 118, 145, 166
....175, 263, 318
SQLCAXGRP...**190**-191
SQLCCSID..**181**, 192, 194
SQLCIGRP...**169**, 179
SQLCINRD..**145**, 164
SQLCIROW **164**, 169
SQLCNAME...**179**
SQLCOMMENTS...**192**, 194
SQLDA **167**, 176, 198
SQLDAGRP..**176**, 192, 194
SQLDARD **108**-110, 112, 145, 167
SQLDAROW **167**, 176
SQLDIAGNAME..**181**
SQLDTA.......................................**78**, 80, 85-86, 97-98
....139, 143, 145, 155-156
SQLDTAGRP **156**-157, 159
SQLDTAMRW **145**, 155
SQLDTARD **97**, 99-100, 145, 153
SQLERR..**437**, 440
SQLERRD...**191**
SQLERRMSG...**191**
SQLERRRM **287**, 331
SQLLABEL...**192**, 194
SQLLENGTH **181**, 192, 194
SQLNAME **181**, 192, 194
SQLNUM..**187**
SQLNUMGRP...**173**, 187
SQLNUMROW **163**-166, 173
SQLOBJECTNAME...**186**
SQLOBJGRP **186**
SQLOBJNAM...**112**, 145, 172
SQLPA...**166**
SQLPAGRP..**174**, 188
SQLPALS...**188**
SQLPARD **145**, 166
SQLPAROW **166**, 174
SQLPCOL...**188**
SQLPOBJ **188**
SQLPOTP **188**
SQLPOWN...**188**
SQLPPRVS **188**
SQLPRECISION **181**, 192, 194
SQLRDBNAME...**191**
SQLRSGRP **168**, 177
SQLRSLOCATOR..**177**
SQLRSLRD **100**, 145, 163
SQLRSNAME...**177**
SQLRSROW...**163**, 168
SQLSCALE...**181**, 192, 194
SQLSTATE **64**, 83, 122, 282-283
....285, 288, 290, 331, 437
SQLSTATEMENT **185**
SQLSTT **67**, 70, 108-109, 114, 145, 171, 251, 260
SQLSTTGRP **171**, 185
SQLSTTVRB....................................**67**, 145, 165, 251
SQLTYPE **181**, 192, 194
SQLUDTGRP..**196**
SQLUDTNAME...**196**
SQLVRBGRP **170**, 181
SQLVRBROW...**165**, 170
SQLWARN..**191**
SRRBACK...**136**
SRRCMIT **131**
SSCP 512
ST rule **302**
staging area **8**
standard focal point messages............................**433**
standard query blocks 310
standardized object name.....................................**278**
statement execution flow.....................**405**-406, 427
statement execution logical flow...........................**31**
statement group description................................**185**
statement row description...................................**171**
statement variables group **181**
statement variables row description **170**
static commit **133**
static commit and rollback **116**
static rollback **130**, 133
stored procedure DDM flow **119**
stored procedure name **277**
stored procedures **7**
stored procedures rule (SP rule)..........................**301**
Structured Query Language...................................**19**
subfield **360**

subvector **360**
supervisor **42**, 46
SV rule **303**
sync point manager **7**, 22, 43
.... 56, 116, 120, 134, 136, 408-409, 413
synchronization point **32**, 387
synchronization point (sync point) 512
synchronization point manager 512
synchronous wait protocol verbs **395**
SYNCMNT **408**
SYNCPTMGR **22**, 43, 116, 120, 136, 408-409, 413
SYNCPTOV **7**
SYNC_LEVEL **392**, 407, 413, 415
SYNERRCD **287**
synonym **278**
SYNTAX alert **374**
SYNTAXRM **287**, 303, 331
system services control point (SSCP) 513
Systems Network Architecture **385**
Systems Network Architecture (SNA) 513
TAKE_BACKOUT **136**
TAKE_SYNCPT **123**
target program **279**
target program name **308**
target program name (TPN) 513
target program name rule (TPN rule) **308**
TCP/IP **421**-422, 513
TCP/IP and DRDA **424**
TCP/IP communications **6**
TCP/IP communications manager **41**, 421
TCP/IP connection **424**, 436
TCP/IP connection rule usage **434**
TCP/IP correlation value display **433**-434
TCP/IP environment usage in DRDA **433**
TCP/IP flow **427**, 429
TCP/IP initialization flow **425**
TCP/IP initialization processing **424**
TCP/IP packet flow **426**
TCPCMNI **424**
TCPCMNT **428**
TCPSRCCD **426**
TCPSRCCR **426**
terminating conversations **407**-409
terminating network connection **428**-429
termination **35**
termination flows **34**
termination, interruption, continuation **318**
time description **217**
timely communication outage notification **341**
timely failure notification **387**
timestamp description **218**
token **415**
tool and program **414**
TP 513
TPN **26**, 40-41, 308, 513
transaction 513
transaction processing **342**
transaction program (TP) 513
transaction program name . **26**, 40-41, 418-419, 513
transparency **48**
transport control protocol **422**
transport control protocol/internet protocol ... **421**
TRGNSPRM **331**
triplet **138**, 141, 149, 157
.... 197, 243, 250-251, 253, 513
triplet override **200**
two-byte description **202**
two-phase commit **32**, 43, 116
.... 120, 131, 134, 305, 407-409, 413
two-phase commit protocols 513
TYPDEFNAM **54**, 67, 74, 145
.... 162, 198, 250, 252, 254, 282, 290
TYPDEFOVR **54**, 67, 97, 100
.... 162, 197, 252, 254, 282, 291
type-independent verb **388**, 390
unique identifier **414**
unit of work **37**, 116-117
.... 120-121, 134, 282, 304, 414, 513
Universal Unique Identifier (UUID) 514
unlike 514
unprotected conversation 514
unprotected network connection 514
UOWID (unit of work identifier) **26**
UP rule **305**
update control rule (UP rule) **305**
update privilege **116**, 120
usage of names **417**
user causes **359**
user ID verification **390**
user-defined data group **196**
userid and encrypted password **349**
Userid and Password **61**, 64
userid and password **345**
userid and password substitute **348**
Userid Only **61**, 64
userid security **345**
userid, password, new password **346**
userid-only **347**
USER_ID **270**, 393
UUID 514
VALNSPRM **80**, 86, 282, 291, 295, 331
variable byte description **220**

variable character mixed description.................**231**
variable character SBCS description...................**225**
variable-character description **228**
verb...**419**
verb categories **388**
verb, LU 6.2..**388**
version ID...**275**
warning...**437**
well known port..**41**
well-known port 514
WN rule **306**
WRITE socket call..**427**
zoned decimal description **208**