

 *Technical Standard*

**DRDA, Version 4, Volume 1:**

**Distributed Relational Database Architecture (DRDA)**

*The Open Group*



© February 2007, The Open Group

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the copyright owners.

This documentation and the software to which it relates are derived in part from copyrighted materials supplied by International Business Machines. Neither International Business Machines nor The Open Group makes any warranty of any kind with regard to this material, including but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Open Group shall not be liable for errors contained herein, or for any direct or indirect, incidental, special, or consequential damages in connection with the furnishing, performance, or use of this material.

Technical Standard

DRDA, Version 4, Volume 1: Distributed Relational Database Architecture (DRDA)

ISBN: 1-931624-70-4

Document Number: C066

Published in the U.K. by The Open Group, February 2007.

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

The Open Group  
Thames Tower  
37-45 Station Road  
Reading  
Berkshire, RG1 1LX  
United Kingdom

or by Electronic Mail to:

[OGSpecs@opengroup.org](mailto:OGSpecs@opengroup.org)

# Contents

<b>Chapter</b>	<b>1</b>	<b>The DRDA Specification .....</b>	<b>1</b>
	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 6.....	3
	1.1.3	What it Means to Implement DRDA Level 5.....	9
	1.1.4	What it Means to Implement DRDA Level 4.....	29
	1.1.5	What it Means to Implement DRDA Level 3.....	32
	1.1.6	What it Means to Implement DRDA Distributed Unit of Work.....	35
	1.2	The FD:OCA Reference .....	38
	1.3	The DDM Reference.....	39
<b>Part</b>	<b>1</b>	<b>Database Access Protocol .....</b>	<b>43</b>
<b>Chapter</b>	<b>2</b>	<b>Introduction to DRDA.....</b>	<b>45</b>
	2.1	DRDA Structure and Other Architectures.....	45
	2.2	DRDA and SQL.....	45
	2.3	DRDA Connection Architecture .....	46
	2.4	Types of Distribution .....	46
	2.5	DRDA Protocols and Functions .....	49
<b>Chapter</b>	<b>3</b>	<b>Using DRDA: Overall Flows.....</b>	<b>53</b>
	3.1	Introduction to Protocol Flows .....	53
	3.1.1	Initialization Flows .....	54
	3.1.2	Bind Flows.....	57
	3.1.3	SQL Statement Execution Flows.....	59
	3.1.4	Commit Flows .....	61
	3.1.5	Termination Flows .....	63
	3.1.6	Utility Flows.....	65
	3.1.6.1	Packet Flow .....	65
<b>Chapter</b>	<b>4</b>	<b>The DRDA Processing Model and Command Flows .....</b>	<b>67</b>
	4.1	DDM and the Processing Model.....	68
	4.2	DRDA Relationship to DDM.....	69
	4.3	The DRDA Processing Model .....	70
	4.3.1	DRDA Managers .....	70
	4.3.1.1	SNA Communications Manager.....	70
	4.3.1.2	SNA Sync Point Communications Manager.....	71
	4.3.1.3	TCP/IP Communications Manager .....	71
	4.3.1.4	Agent.....	71
	4.3.1.5	Supervisor .....	72
	4.3.1.6	Security Manager .....	72
	4.3.1.7	Directory .....	73

4.3.1.8	Dictionary .....	73
4.3.1.9	Resynchronization Manager.....	73
4.3.1.10	Sync Point Manager .....	73
4.3.1.11	SQL Application Manager .....	74
4.3.1.12	Relational Database Manager.....	76
4.3.1.13	CCSID Manager.....	77
4.3.1.14	XA Manager .....	77
4.3.2	The DRDA Processing Model Flow.....	78
4.3.3	Product-Unique Extensions.....	84
4.3.4	Diagnostic and Problem Determination Support in DRDA .....	84
4.3.5	Intermediate Server Processing.....	85
4.3.5.1	Overview and Terminology.....	85
4.3.5.2	Examples .....	86
4.4	DDM Commands and Replies .....	88
4.4.1	Accessing a Remote Relational Database Manager .....	89
4.4.2	DRDA Security Flows .....	97
4.4.2.1	Identification and Authentication Security Flows .....	97
4.4.2.2	Security-Sensitive Data Encryption Security Flow .....	111
4.4.2.3	Intermediate Server Processing Security Flow for Security-Sensitive Data Encryption.....	117
4.4.2.4	Trusted Application Server.....	119
4.4.2.5	Establishing a Trusted Connection .....	119
4.4.2.6	Switch ID Associated with a Trusted Connection.....	123
4.4.3	Performing the Bind Operation and Creating a Package.....	127
4.4.3.1	Perform the Bind Copy Operation to Copy an Existing Package.....	132
4.4.3.2	Perform the Bind Deploy Operation to Deploy an Existing Package.....	136
4.4.4	Deleting an Existing Package .....	139
4.4.4.1	Delete Many Existing Packages .....	141
4.4.5	Performing a Rebind Operation.....	143
4.4.6	Activating and Processing Queries .....	145
4.4.6.1	Fixed Row Protocol.....	148
4.4.6.2	Limited Block Protocol (No FD:OCA Generalized String Data in Answer Set).....	157
4.4.6.3	Limited Block Protocol (FD:OCA Generalized String Data in Answer Set) .....	164
4.4.7	Executing a Bound SQL Statement.....	171
4.4.7.1	Executing Ordinary Bound SQL Statements.....	171
4.4.7.2	Invoking a Stored Procedure that Returns Result Sets .....	177
4.4.7.3	Executing Chained Ordinary Bound SQL Statements as an Atomic Operation .....	188
4.4.7.4	Executing Bound SQL Statement with Array Input .....	193
4.4.8	Preparing an SQL Statement .....	195
4.4.9	Retrieving the Data Variable Definitions of an SQL Statement .....	198
4.4.10	Executing a Describe Table SQL Statement.....	200
4.4.11	Executing a Dynamic SQL Statement .....	202
4.4.12	Returning SQL Diagnostics .....	204
4.4.13	Controlling the Amount of Describe Information Returned .....	208
4.4.14	Interrupting a Running DRDA Request .....	209
4.4.15	Commitment of Work in DRDA .....	212

4.4.15.1	Commitment of Work in a Remote Unit of Work.....	214
4.4.15.2	Commitment of Work in a Distributed Unit of Work.....	217
4.4.15.3	Global and Local Transactions .....	235
4.4.16	Connection Reuse.....	239
4.4.16.1	Connection Pooling.....	239
4.4.16.2	Transaction Pooling .....	240
<b>Chapter 5</b>	<b>Data Definition and Exchange.....</b>	<b>247</b>
5.1	Use of FD:OCA .....	247
5.2	Use of Base and Option Sets.....	248
5.2.1	Basic FD:OCA Object Contained in DDM.....	248
5.2.2	DRDA FD:OCA Object.....	250
5.2.3	Early and Late Descriptors .....	253
5.3	Relationship of DRDA and DDM Objects and Commands.....	256
5.3.1	DRDA Command to Descriptor Relationship .....	256
5.3.2	Descriptor Classes.....	259
5.4	DRDA Descriptor Definitions .....	262
5.5	Late Descriptors .....	263
5.5.1	Late Array Descriptors.....	263
5.5.1.1	SQLDTARD: SQL Communication Area with Data Array Description .....	264
5.5.1.2	SQLDTAMRW: Data Array Description for Multi-Row Input .....	265
5.5.2	Late Row Descriptors .....	266
5.5.2.1	SQLDTA: Data Description for One Row of Data.....	267
5.5.2.2	SQLCADTA: Data Description for One Row with SQL Communication Area and Data .....	268
5.5.3	Late Group Descriptors.....	269
5.5.3.1	SQLDTAGRP: Data Descriptions for One Row of Data .....	271
5.5.3.2	Overriding Output Formats .....	274
5.6	Early Descriptors.....	277
5.6.1	Initial DRDA Type Representation.....	277
5.6.2	Early Array Descriptors .....	277
5.6.2.1	SQLRSLRD: Data Array of a Result Set.....	278
5.6.2.2	SQLCINRD: SQL Result Set Column Array Description.....	279
5.6.2.3	SQLSTTVRB: SQL Statement Variable Description .....	280
5.6.2.4	SQLDARD: SQL Descriptor Area Row Description with SQL Communication Area.....	281
5.6.2.5	SQLDCTOKS: SQL Diagnostics Condition Token Array.....	282
5.6.2.6	SQLDIAGCI: SQL Diagnostics Condition Information Array .....	283
5.6.2.7	SQLDIAGCN: SQL Diagnostics Connection Array .....	284
5.6.3	Early Row Descriptors.....	285
5.6.3.1	SQLRSROW: SQL Row Description of One Result Set Row .....	285
5.6.3.2	SQLVRBROW: SQL Statement Variable Row Description.....	286
5.6.3.3	SQLSTT: SQL Statement Row Description.....	287
5.6.3.4	SQLOBJNAM: SQL Object Name Row Description .....	288
5.6.3.5	SQLNUMROW: SQL Number of Elements Row Description.....	289
5.6.3.6	SQLCARD: SQL Communication Area Row Description .....	290
5.6.3.7	SQLDAROW: SQL Descriptor Area Row Description .....	291
5.6.3.8	SQLDHROW: SQL Descriptor Header Row Description .....	292

5.6.3.9	SQLCNROW: SQL Diagnostics Connection Row .....	293
5.6.3.10	SQLDCROW: SQL Diagnostics Condition Row .....	294
5.6.3.11	SQLTOKROW: SQL Diagnostics Token Row .....	295
5.6.4	Early Group Descriptors .....	296
5.6.4.1	SQLRSGRP: SQL Result Set Group Description.....	296
5.6.4.2	SQLVRBGRP: SQL Statement Variable Group Description .....	297
5.6.4.3	SQLSTTGRP: SQL Statement or Attributes Group Description .....	299
5.6.4.4	SQLOBJGRP: SQL Object Name Group Description .....	300
5.6.4.5	SQLNUMGRP: SQL Number of Elements Group Description .....	301
5.6.4.6	SQLCAGRP: SQL Communication Area Group Description .....	302
5.6.4.7	SQLCAXGRP: SQL Communication Area Exceptions Group Description .....	303
5.6.4.8	SQLDIAGGRP: SQL Diagnostics Group Description.....	305
5.6.4.9	SQLDAGRP: SQL Descriptor Area Group Description .....	306
5.6.4.10	SQLUDTGRP: SQL Descriptor User-Defined Type Group Description .....	308
5.6.4.11	SQLDHGRP: SQL Descriptor Header Group Description .....	310
5.6.4.12	SQLDOPTGRP: SQL Descriptor Optional Group Description .....	313
5.6.4.13	SQLDXGRP: SQL Descriptor Extended Group Description .....	315
5.6.4.14	SQLNUMEXT: SQL Extent Description for Variable Arrays .....	317
5.6.4.15	SQLEXTROW: SQL Array Row Description for a Variable Array.....	318
5.6.4.16	SQLEXTGRP: SQL Extent Group Description for a Variable Array.....	319
5.6.4.17	SQLDIAGSTT: SQL Diagnostics Statement Group Description .....	320
5.6.4.18	SQLCNGRP: SQL Diagnostics Connection Group Description .....	324
5.6.4.19	SQLDCGRP: SQL Diagnostics Condition Group Description .....	326
5.6.4.20	SQLDCXGRP: SQL Diagnostics Extended Names Group Description .....	330
5.6.4.21	SQLTOKGRP: SQL Diagnostics Token Group .....	332
5.6.5	Early Environmental Descriptors .....	333
5.6.5.1	Four-Byte Integer .....	337
5.6.5.2	Two-Byte Integer .....	338
5.6.5.3	One-Byte Integer .....	339
5.6.5.4	Sixteen-Byte Basic Float.....	340
5.6.5.5	Eight-Byte Basic Float .....	341
5.6.5.6	Four-Byte Basic Float .....	342
5.6.5.7	Fixed Decimal .....	343
5.6.5.8	Zoned Decimal .....	344
5.6.5.9	Numeric Character.....	345
5.6.5.10	Result Set Locator.....	346

5.6.5.11	Eight-Byte Integer .....	347
5.6.5.12	Large Object Bytes Locator .....	348
5.6.5.13	Large Object Character Locator .....	349
5.6.5.14	Large Object Character DBCS Locator .....	350
5.6.5.15	Row Identifier .....	351
5.6.5.16	Date .....	352
5.6.5.17	Time .....	353
5.6.5.18	Timestamp .....	354
5.6.5.19	Fixed Bytes .....	355
5.6.5.20	Variable Bytes .....	356
5.6.5.21	Long Variable Bytes .....	357
5.6.5.22	Null-Terminated Bytes .....	358
5.6.5.23	Null-Terminated SBCS .....	359
5.6.5.24	Fixed Character SBCS .....	360
5.6.5.25	Variable Character SBCS .....	361
5.6.5.26	Long Variable Character SBCS .....	362
5.6.5.27	Fixed-Character DBCS (GRAPHIC) .....	363
5.6.5.28	Variable-Character DBCS (GRAPHIC) .....	364
5.6.5.29	Long Variable Character DBCS (GRAPHIC) .....	365
5.6.5.30	Fixed Character Mixed .....	366
5.6.5.31	Variable Character Mixed .....	367
5.6.5.32	Long Variable Character Mixed .....	368
5.6.5.33	Null-Terminated Mixed .....	369
5.6.5.34	Pascal L String Bytes .....	370
5.6.5.35	Pascal L String SBCS .....	371
5.6.5.36	Pascal L String Mixed .....	372
5.6.5.37	SBCS Datalink .....	373
5.6.5.38	Mixed-Byte Datalink .....	374
5.6.5.39	Decimal Floating Point .....	375
5.6.5.40	Boolean .....	376
5.6.5.41	Fixed Binary .....	377
5.6.5.42	Variable Binary .....	378
5.6.5.43	XML String Internal Encoding .....	379
5.6.5.44	XML String External Encoding .....	380
5.6.5.45	Large Object Bytes .....	381
5.6.5.46	Large Object Character SBCS .....	382
5.6.5.47	Large Object Character DBCS (GRAPHIC) .....	383
5.6.5.48	Large Object Character Mixed .....	384
5.6.6	Late Environmental Descriptors .....	385
5.7	FD:OCA Meta Data Summary .....	387
5.7.1	Overriding Descriptors to Handle Problem Data .....	391
5.7.1.1	Overriding Everything .....	391
5.7.1.2	Overriding Some User Data .....	392
5.7.1.3	Assigning LIDs to O Triplets .....	393
5.7.2	MDD Materialization Rules .....	394
5.7.3	Error Checking and Reporting for Descriptors .....	394
5.7.3.1	General Errors .....	395
5.7.3.2	MDD Errors .....	395
5.7.3.3	SDA Errors .....	395
5.7.3.4	GDA/CPT Errors .....	395
5.7.3.5	RLO Errors .....	396
5.8	DRDA Examples .....	397

5.8.1	Environmental Description Objects .....	397
5.8.1.1	Late Environmental Descriptors .....	397
5.8.1.2	Early Data Unit Descriptors .....	399
5.8.1.3	Late Data Unit Descriptors .....	401
5.8.2	Command Execution Examples .....	401
5.8.2.1	EXECUTE IMMEDIATE .....	402
5.8.2.2	Open Query Statement .....	402
5.8.2.3	Input Variable Arrays SQL Request .....	406
5.8.2.4	Call (Stored Procedure) .....	407
5.8.2.5	Call (Stored Procedure Returning Result Sets) .....	409
5.8.2.6	Enabling Dynamic Data Format .....	411
<b>Chapter 6</b>	<b>Names .....</b>	<b>415</b>
6.1	End Users .....	416
6.1.1	Support for End-User Names .....	416
6.2	RDBs .....	417
6.3	Tables and Views .....	418
6.4	Packages .....	419
6.4.1	Package Name .....	419
6.4.2	Package Consistency Token .....	420
6.4.3	Package Version ID .....	420
6.4.4	Command Source Identifiers .....	421
6.4.5	Package Collection Resolution .....	421
6.5	Stored Procedure Names .....	423
6.6	Synonyms and Aliases .....	424
6.7	Default Mechanisms for Standardized Object Names .....	424
6.8	Target Program .....	425
<b>Chapter 7</b>	<b>DRDA Rules .....</b>	<b>427</b>
7.1	Atomic Chaining (AC Rules) .....	427
7.2	Connection Allocation (CA Rules) .....	429
7.3	Mapping of Reply Messages to SQLSTATEs (CD Rules) .....	431
7.4	Connection Failure (CF Rules) .....	431
7.5	Commit/Rollback Processing (CR Rules) .....	432
7.6	Connection Usage (CU Rules) .....	436
7.7	Conversion of Data Types (DC Rules) .....	440
7.8	Data Format (DF Rules) .....	444
7.9	Data Representation Transformation (DT Rules) .....	448
7.10	RDB-Initiated Rollback (IR Rules) .....	452
7.11	Optionality (OC Rules) .....	453
7.12	Program Binding (PB Rules) .....	454
7.13	SQL Diagnostics (SD Rules) .....	457
7.14	Security (SE Rules) .....	458
7.15	SQL Section Number Assignment (SN Rules) .....	460
7.16	Stored Procedures (SP Rules) .....	463
7.17	SET Statement (ST Rules) .....	464
7.18	Serviceability (SV Rules) .....	465
7.19	Update Control (UP Rules) .....	467
7.20	Passing Warnings to the Application Requester (WN Rules) .....	468
7.21	Names .....	469
7.21.1	End-User Names (EUN Rules) .....	469

	7.21.2	SQL Object Names (ON Rules) .....	469
	7.21.3	Relational Database Names (RN Rules) .....	470
	7.21.4	Target Program Names (TPN Rules).....	471
	7.22	Query Processing .....	473
	7.22.1	Blocking .....	473
	7.22.1.1	Block Formats (BF Rules) .....	474
	7.22.1.2	Block Size (BS Rules).....	477
	7.22.1.3	Chaining (CH Rules) .....	478
	7.22.2	Query Instances (QI Rules).....	482
	7.22.3	Query Data Transfer Protocols (QP Rules).....	484
	7.22.4	Query Data or Result Set Transfer (QT Rules) .....	490
	7.22.5	Additional Query and Result Set Termination Rules .....	491
	7.22.5.1	Rules for OPNQRY, CNTQRY, CLSQRY, and EXCSQLSTT .....	492
	7.22.5.2	Rules for FETCH .....	497
	7.22.5.3	Rules for CLOSE.....	500
<b>Chapter</b>	<b>8</b>	<b>SQLSTATE Usage .....</b>	<b>503</b>
	8.1	DRDA Reply Messages and SQLSTATE Mappings.....	503
	8.2	SQLSTATE Codes Referenced by DRDA.....	505
<b>Part</b>	<b>2</b>	<b>Environmental Support .....</b>	<b>511</b>
<b>Chapter</b>	<b>9</b>	<b>Environmental Support.....</b>	<b>513</b>
	9.1	DDM Communications Model and Network Protocol Support .....	513
	9.2	Accounting .....	514
	9.3	Transaction Processing .....	514
<b>Chapter</b>	<b>10</b>	<b>Security .....</b>	<b>515</b>
	10.1	DCE Security Mechanisms with GSS-API.....	515
	10.2	User ID-Related Security Mechanisms .....	516
	10.2.1	User ID and Password.....	517
	10.2.2	User ID, Password, and New Password .....	518
	10.2.3	User ID-Only.....	519
	10.2.4	User ID and Original or Strong Password Substitute.....	520
	10.2.5	User ID and Encrypted Password .....	521
	10.2.6	Encrypted User ID and Password .....	522
	10.2.7	Encrypted User ID, Password, and New Password .....	523
	10.3	Kerberos.....	525
	10.3.1	Kerberos Protocol .....	525
	10.3.2	Kerberos Security Mechanism with SSPI and GSS-API .....	525
	10.3.3	Trusted Application Server.....	527
	10.4	User ID and Data-Related Security Mechanisms .....	528
	10.4.1	Encrypted User ID and Security-Sensitive Data.....	528
	10.4.2	Encrypted User ID, Password, and Security-Sensitive Data .....	532
	10.4.3	Encrypted User ID, Password, New Password, and Security-Sensitive Data .....	534
	10.5	Plug-In Security Mechanism .....	536

<b>Chapter</b>	<b>11</b>	<b>Problem Determination .....</b>	<b>539</b>
	11.1	Network Management Tools and Techniques .....	539
	11.1.1	Standard Focal Point Messages.....	539
	11.1.2	Focal Point.....	539
	11.1.3	Correlation .....	540
	11.2	Monitoring .....	541
	11.2.1	Verification of Network Connectivity Flag .....	541
	11.2.2	Request and Response Packet Object.....	541
	11.2.3	Elapsed Time.....	542
	11.2.4	Ping .....	542
	11.3	DRDA Required Problem Determination and Isolation Enhancements.....	543
	11.3.1	Correlation Displays.....	543
	11.3.2	DRDA Diagnostic Information Collection and Correlation .....	543
	11.3.2.1	Data Collection .....	543
	11.3.2.2	Correlation Between Focal Point Messages and Supporting Data .....	544
	11.4	Generic Focal Point Messages and Message Models.....	545
	11.4.1	When to Generate Alerts.....	545
	11.4.2	Alerts and Alert Structure.....	545
	11.4.2.1	Alert Implementation Basics .....	545
	11.4.3	Error Condition to Alert Model Mapping.....	546
	11.4.3.1	Specific Alert to DDM Reply Message Mapping.....	546
	11.4.3.2	Additional Alerts at the Application Requester .....	548
	11.4.3.3	DRDA-Defined Alert Models.....	548
	11.4.4	Alert Example .....	566
	11.4.4.1	Major Vector/Subvector/Subfield Construction.....	566
<b>Part</b>	<b>3</b>	<b>Network Protocols .....</b>	<b>573</b>
<b>Chapter</b>	<b>12</b>	<b>SNA.....</b>	<b>575</b>
	12.1	SNA and the DDM Communications Model.....	575
	12.2	What You Need to Know About SNA and LU 6.2 .....	575
	12.3	LU 6.2 .....	575
	12.4	LU 6.2 Verb Categories .....	576
	12.5	LU 6.2 Product-Support Subsetting.....	577
	12.6	LU 6.2 Base and Option Sets.....	577
	12.6.1	Base Set Functions.....	577
	12.6.1.1	Basic Conversation Verb Category .....	577
	12.6.1.2	Type-Independent Verb Category .....	577
	12.6.2	Option Set Functions .....	578
	12.6.2.1	Basic Conversation Verb Category .....	578
	12.6.2.2	Type-Independent Verb Category .....	578
	12.7	LU 6.2 and DRDA .....	580
	12.7.1	Initializing a Conversation .....	580
	12.7.1.1	LU 6.2 Verbs that the Application Requester Uses.....	580
	12.7.1.2	LU 6.2 Verbs that the Application Server Uses.....	582
	12.7.1.3	Initialization Flows .....	583
	12.7.2	Processing a DRDA Request.....	589
	12.7.2.1	LU 6.2 Verbs that the Application Requester Uses.....	589

12.7.2.2	LU 6.2 Verbs that the Application Server Uses .....	590
12.7.2.3	Bind Flows.....	590
12.7.2.4	SQL Statement Execution Flows .....	593
12.7.3	Terminating a Conversation .....	596
12.7.3.1	LU 6.2 Verbs that the Application Requester Uses.....	596
12.7.3.2	LU 6.2 Verbs that the Application Server Uses .....	597
12.7.3.3	Termination Flow: SYNC_LEVEL(NONE) Conversation.....	597
12.7.3.4	Termination Flow: SYNC_LEVEL(SYNCPT) Conversation.....	597
12.7.4	Commit Flows on SYNC_LEVEL(NONE) Conversations .....	599
12.7.5	Rollback Flows on SYNC_LEVEL(NONE) Conversations .....	600
12.7.6	Commit Flows on SYNC_LEVEL(SYNCPT) Conversations .....	600
12.7.7	Rollback Flows on SYNC_LEVEL(SYNCPT) Conversations .....	601
12.7.8	Handling Conversation Failures.....	602
12.7.9	Managing Conversations Using Distributed Unit of Work.....	602
12.8	SNA Environment Usage in DRDA .....	603
12.8.1	Problem Determination in SNA Environments.....	603
12.8.1.1	LUWID.....	603
12.8.1.2	DRDA LUWID and Correlation of Diagnostic Information .....	603
12.8.1.3	Data Collection .....	604
12.8.1.4	Alerts and Supporting Data in SNA Environments .....	604
12.8.2	Rules Usage for SNA Environments .....	605
12.8.2.1	LU 6.2 Usage of Connection Allocation Rules.....	605
12.8.2.2	LU 6.2 Usage of Commit/Rollback Processing Rules .....	606
12.8.2.3	LU 6.2 Usage of Security (SE Rules).....	607
12.8.2.4	LU 6.2 Usage of Serviceability Rules.....	608
12.8.2.5	LU 6.2 Usage of Names .....	609
12.8.3	Transaction Program Names .....	610
<b>Chapter 13</b>	<b>TCP/IP .....</b>	<b>611</b>
13.1	TCP/IP and the DDM Communications Model .....	611
13.2	What You Need to Know About TCP/IP .....	611
13.3	TCP/IP.....	612
13.3.1	Transport Control Protocol (TCP).....	612
13.3.2	Application Services .....	613
13.4	Sockets Interface.....	613
13.5	TCP/IP and DRDA .....	614
13.5.1	Initializing a Connection.....	614
13.5.1.1	Initialization Flows .....	614
13.5.2	Processing a DRDA Request.....	616
13.5.2.1	Bind Flows.....	616
13.5.2.2	SQL Statement Execution Flows .....	617
13.5.3	Terminating a Connection.....	618
13.5.4	Commit Flows .....	619
13.5.4.1	Remote Unit of Work .....	619

13.5.4.2	Distributed Unit of Work Using DDM Sync Point Manager .....	620
13.5.5	Handling Connection Failures .....	622
13.6	TCP/IP Environment Usage in DRDA .....	623
13.6.1	Problem Determination in TCP/IP Environments .....	623
13.6.1.1	Standard Focal Point Messages.....	623
13.6.1.2	Focal Point Support .....	623
13.6.1.3	Correlation and Correlation Display.....	623
13.6.2	Rules Usage for TCP/IP Environments.....	624
13.6.2.1	TCP/IP Usage of Connection Allocation Rules.....	624
13.6.2.2	TCP/IP Usage of Commit/Rollback Processing Rules .....	625
13.6.2.3	TCP/IP Usage of Security (SE Rules).....	625
13.6.2.4	TCP/IP Usage of Serviceability Rules .....	625
13.6.2.5	TCP/IP Usage of Relational Database Names Rules.....	626
13.6.2.6	TCP/IP Usage of PORT for DRDA Service Rules .....	626
13.6.3	Service Names .....	626
<b>Appendix A</b>	<b>DDM Managers, Commands, and Reply Messages .....</b>	<b>629</b>
A.1	DDM Manager Relationship to DRDA Functions.....	629
A.2	DDM Commands and Reply Messages .....	630
<b>Appendix B</b>	<b>Scrollable Cursor Overview .....</b>	<b>707</b>
B.1	Key Definitions.....	707
B.2	Attributes.....	707
B.2.1	Scrollability Attribute .....	707
B.2.2	Sensitivity Attribute.....	708
B.2.3	Updatability Attribute.....	709
B.3	Operations.....	709
B.4	Choice of Query Protocol.....	710
B.5	DRDA Rowsets.....	711
B.6	Cursor Position Management .....	712
B.7	Cursor Position Rules .....	713
B.8	Cursor Disposition.....	714
B.9	Scrolling for Stored Procedure Result Sets .....	714
B.10	Downlevel Requesters.....	715
B.11	Intermediate Data Server Processing .....	715
B.11.1	Example: qryrstblk Not Specified.....	715
B.11.2	Example: qryrstblk Set to TRUE .....	716
<b>Appendix C</b>	<b>Rowset Processing.....</b>	<b>719</b>
C.1	Rowset Cursors .....	719
C.2	SQL Rowsets .....	719
<b>Appendix D</b>	<b>SQL Function Codes .....</b>	<b>721</b>
<b>Appendix E</b>	<b>Failover Overview.....</b>	<b>723</b>
	<b>Glossary .....</b>	<b>725</b>
	<b>Index.....</b>	<b>735</b>

**List of Figures**

2-1	Degrees of Distribution of Database Function.....	47
2-2	DRDA Network.....	50
2-3	DRDA Network Implementation Example.....	50
3-1	Logical Flow: Initialization Flows with SNA Security .....	55
3-2	Logical Flow: Initialization Flows with DCE Security .....	56
3-3	Logical Flow: Bind Flows.....	57
3-4	Logical Flow: SQL Statement Execution Flows .....	60
3-5	Logical Flow: DRDA Two-Phase Commit.....	62
3-6	Logical Flow: DRDA One-Phase Commit Using DDM Commands ..	62
3-7	Logical Flow: DRDA Two-Phase Commit Termination Flows Using DDM Commands .....	63
3-8	Logical Flow: SNA Termination Flows on Protected Conversations ..	64
3-9	Utility Flow: DRDA Packet Flows Using DDM Commands.....	65
4-1	DRDA Processing Model .....	79
4-2	Establishing a Connection to a Remote Database Manager .....	89
4-3	Kerberos or DCE Security Flow .....	98
4-4	Encryption and Substitution Flow.....	101
4-5	Plug-In Security Flows .....	105
4-6	N-Flow Security Authentication.....	108
4-7	Security-Sensitive Data Encryption: Example for Requester Server Processing .....	111
4-8	Security-Sensitive Data Encryption: Intermediate Server Processing Using SECTKNOVR .....	113
4-9	Security-Sensitive Data Encryption: Example for Intermediate Server Processing .....	115
4-10	Intermediate Server Security-Sensitive Data Encryption and Decryption.....	117
4-11	Establishing a Trusted Connection .....	120
4-12	Switching ID Associated with a Trusted Connection .....	124
4-13	Binding and/or Package Creation (Part 1).....	127
4-14	Binding and/or Package Creation (Part 2).....	128
4-15	Performing the Bind Copy Operation to Copy an Existing Package .....	133
4-16	Performing the Bind Deploy Operation to Deploy an Existing Package .....	136
4-17	Dropping an Existing Package .....	139
4-18	Deleting Many Existing Packages .....	141
4-19	Rebinding an Existing Package.....	143
4-20	Fixed Row Protocol Query Processing (Part 1).....	148
4-21	Fixed Row Protocol Query Processing (Part 2).....	149
4-22	Limited Block Protocol Query Processing (No FD:OCA Generalized String Data) (Part 1) .....	157
4-23	Limited Block Protocol Query Processing (No FD:OCA Generalized String Data) (Part 2) .....	158
4-24	Limited Block Protocol Query Processing (FD:OCA Generalized String Data, rtnextall) (Part 1) .....	165
4-25	Limited Block Protocol Query Processing (FD:OCA Generalized String Data, rtnextall) (Part 2) .....	166
4-26	Limited Block Protocol Query Processing (Externalized Data, rtnextrow).....	169

4-27	Executing a Bound SQL Statement.....	172
4-28	Executing a Stored Procedure (Part 1) .....	178
4-29	Executing a Stored Procedure (Part 2) .....	179
4-30	Executing a Stored Procedure (Part 3) .....	180
4-31	Executing Bound SQL Statements as an Atomic Operation (Part 1)...	189
4-32	Executing Bound SQL Statements as an Atomic Operation (Part 2)...	190
4-33	Executing an Array Input SQL Statement.....	193
4-34	Preparing an SQL Statement .....	195
4-35	Describing a Bound SQL Statement .....	198
4-36	Describing a Table .....	200
4-37	Immediate Execution of SQL Work.....	202
4-38	Returning SQL Diagnostics .....	205
4-39	Requests to Interrupt DRDA Requests .....	209
4-40	Interrupted DRDA Request.....	210
4-41	Commit a Remote Unit of Work .....	214
4-42	Executing a Bound SQL CALL Statement.....	216
4-43	DRDA Sample Configuration .....	218
4-44	DRDA RDBUPDRM Example Flow.....	219
4-45	Single RDB Update at a DRDA Remote Unit of Work Application Server .....	221
4-46	Single RDB Update Using Distributed Unit of Work .....	225
4-47	Multi-Relational Database Update .....	229
4-48	RDB at AS1 Initiates Rollback .....	233
4-49	RDB at AS2 Initiates Rollback .....	233
4-50	RDB at AS3 Initiates Rollback .....	234
4-51	Reuse Connection using Connection Pooling.....	239
4-52	Reuse DUOW Connection using Transaction Pooling .....	242
4-53	Reuse RUOW Connection using Transaction Pooling.....	244
5-1	Basic FD:OCA Object.....	248
5-2	Basic FD:OCA Object with Externalized Data .....	249
5-3	Conceptual View of a DRDA FD:OCA Object.....	250
5-4	Conceptual View of a DRDA FD:OCA Object with Externalized LOB Columns .....	252
5-5	SQLDTARD Array Descriptor .....	264
5-6	SQLDTAMRW Array Descriptor (Multi-Row Input Data).....	265
5-7	SQLDTA Row Descriptor.....	267
5-8	SQLCADTA Row Descriptor.....	268
5-9	SQLDTAGRP Group Descriptor .....	272
5-10	SQLRSLRD Array Descriptor.....	278
5-11	SQLCINRD Array Descriptor .....	279
5-12	SQLSTTVRB Array Descriptor.....	280
5-13	SQLDARD Array Descriptor.....	281
5-14	SQLDCTOKS Array Descriptor .....	282
5-15	SQLDIAGCI Array Descriptor.....	283
5-16	SQLDIAGCN Array Descriptor.....	284
5-17	SQLRSROW Row Descriptor .....	285
5-18	SQLVRBROW Row Descriptor .....	286
5-19	SQLSTT Row Descriptor (One SQL Statement or SQL Attributes).....	287
5-20	SQLOBJNAM Row Descriptor.....	288
5-21	SQLNUMROW Row Descriptor.....	289
5-22	SQLCARD Row Descriptor .....	290
5-23	SQLDAROW Row Descriptor.....	291

5-24	SQLDHHROW Row Descriptor.....	292
5-25	SQLCNROW Row Descriptor.....	293
5-26	SQLDCROW Row Descriptor.....	294
5-27	SQLTOKROW Row Descriptor.....	295
5-28	SQLRSGRP Group Descriptor.....	296
5-29	SQLVRBGRP Group Descriptor.....	297
5-30	SQLSTTGRP Group Descriptor (One SQL Statement or Attributes) ..	299
5-31	SQLOBJGRP Group Descriptor.....	300
5-32	SQLNUMGRP Group Descriptor.....	301
5-33	SQLCAGRP Group Descriptor.....	302
5-34	SQLCAXGRP Group Descriptor.....	303
5-35	SQLDIAGGRP Group Descriptor.....	305
5-36	SQLDAGRP Group Descriptor.....	306
5-37	SQLUDTGRP Group Descriptor.....	308
5-38	SQLDHGRP Group Descriptor.....	310
5-39	SQLDOPTGRP Group Descriptor.....	313
5-40	SQLDXGRP Group Descriptor.....	315
5-41	SQLNUMEXT Row Descriptor.....	317
5-42	SQLEXTROW Row Descriptor.....	318
5-43	SQLEXTGRP Row Descriptor.....	319
5-44	SQLDIAGSTT Group Descriptor.....	320
5-45	SQLCNGRP Group Descriptor.....	324
5-46	SQLDCGRP Group Descriptor.....	327
5-47	SQLDCXGRP Group Descriptor.....	330
5-48	SQLTOKGRP Group Descriptor.....	332
5-49	DRDA Type X'32,33' SQL Type 448,449 Variable Character SBCS.....	333
5-50	DRDA Type X'02,03' SQL Type 496,497 INTEGER.....	337
5-51	DRDA Type X'04,05' SQL Type 500,501 SMALL INTEGER.....	338
5-52	DRDA Type X'06,07' SQL Type n/a,n/a.....	339
5-53	DRDA Type X'08,09' SQL Type 480,481 FLOAT (16).....	340
5-54	DRDA Type X'0A,0B' SQL Type 480,481 FLOAT (8).....	341
5-55	DRDA Type X'0C,0D' SQL Type 480,481 FLOAT (4).....	342
5-56	DRDA Type X'0E,0F' SQL Type 484,485 FIXED DECIMAL.....	343
5-57	DRDA Type X'10,11' SQL Type 488,489 ZONED DECIMAL.....	344
5-58	DRDA Type X'12,13' SQL Type 504,505 NUMERIC CHARACTER.....	345
5-59	DRDA Type X'14,15' SQL Type 972,973 RESULT SET LOCATOR.....	346
5-60	DRDA Type X'16,17' SQL Type 492,493 EIGHT-BYTE INTEGER.....	347
5-61	DRDA Type X'18,19' SQL Type 960,961 LARGE OBJECT BYTES LOCATOR.....	348
5-62	DRDA Type X'1A,1B' SQL Type 964,965 LARGE OBJ. CHAR. SBCS LOCATOR.....	349

5-63	DRDA Type X'1C,1D' SQL Type 968,969 LARGE OBJ. CHAR. DBCS LOCATOR .....	350
5-64	DRDA Type X'1E,1F' SQL Type 904,905 ROW IDENTIFIER .....	351
5-65	DRDA Type X'20,21' SQL Type 384,385 DATE .....	352
5-66	DRDA Type X'22,23' SQL Type 388,389 TIME .....	353
5-67	DRDA Type X'24,25' SQL Type 392,393 TIMESTAMP .....	354
5-68	DRDA Type X'26,27' SQL Type 452,453 FIXED BYTES .....	355
5-69	DRDA Type X'28,29' SQL Type 448,449 VARIABLE BYTES .....	356
5-70	DRDA Type X'2A,2B' SQL Type 456,457 LONG VAR BYTES .....	357
5-71	DRDA Type X'2C,2D' SQL Type 460,461 NULL-TERMINATED BYTES .....	358
5-72	DRDA Type X'2E,2F' SQL Type 460,461 NULL-TERMINATED SBCS .....	359
5-73	DRDA Type X'30,31' SQL Type 452,453 FIXED CHARACTER SBCS .....	360
5-74	DRDA Type X'32,33' SQL Type 448,449 VARIABLE CHARACTER SBCS .....	361
5-75	DRDA Type X'34,35' SQL Type 456,457 LONG VAR CHARACTER SBCS .....	362
5-76	DRDA Type X'36,37' SQL Type 468,469 FIXED CHARACTER DBCS .....	363
5-77	DRDA Type X'38,39' SQL Type 464,465 VARIABLE CHARACTER DBCS .....	364
5-78	DRDA Type X'3A,3B' SQL Type 472,473 LONG VAR CHARACTER DBCS .....	365
5-79	DRDA Type X'3C,3D' SQL Type 452,453 FIXED CHARACTER MIXED .....	366
5-80	DRDA Type X'3E,3F' SQL Type 448,449 VARIABLE CHARACTER MIXED .....	367
5-81	DRDA Type X'40,41' SQL Type 456,457 LONG VARIABLE CHARACTER MIXED .....	368
5-82	DRDA Type X'42,43' SQL Type 460,461 NULL-TERMINATED MIXED .....	369
5-83	DRDA Type X'44,45' SQL Type 476,477 PASCAL L STRING BYTES .....	370
5-84	DRDA Type X'46,47' SQL Type 476,477 PASCAL L STRING SBCS .....	371
5-85	DRDA Type X'48,49' SQL Type 476,477 PASCAL L STRING MIXED .....	372
5-86	DRDA Type X'4C,4D' SQL Type 396,397 SBCS DATALINK .....	373
5-87	DRDA Type X'4E,4F' SQL Type 396,397 MIXED-BYTE DATALINK .....	374
5-88	DRDA Type X'BA,BB' SQL Type 996,997 DECFLOAT (34) .....	375
5-89	DRDA Type X'BE,BF' SQL Type 2436,2437 BOOLEAN .....	376

5-90	DRDA Type X'C0,C1' SQL Type 912,913 FIXED BINARY .....	377
5-91	DRDA Type X'C2,C3' SQL Type 908,909 VARIABLE BINARY .....	378
5-92	DRDA Type X'C4,C5' SQL Type 988,989 XML .....	379
5-93	DRDA Type X'C6,C7' SQL Type 988,989 XML .....	380
5-94	DRDA Type X'C8,C9' SQL Type 404,405 LARGE OBJECT BYTES .....	381
5-95	DRDA Type X'CA,CB' SQL Type 408,409 LARGE OBJECT CHAR. SBCS.....	382
5-96	DRDA Type X'CC,CD' SQL Type 412,413 LARGE OBJECT CHAR. DBCS.....	383
5-97	DRDA Type X'CE,CF' SQL Type 408,409 LARGE OBJECT CHAR. MIXED.....	384
5-98	DRDA Type X'30', SQL Type 468, MDD Override Example: Base.....	385
5-99	DRDA Type X'30', SQL Type 468, MDD Override Example: Override.....	386
10-1	Using GSS-API to Call DCE-Based Security Flows in DRDA .....	515
10-2	User ID and Password Authentication .....	517
10-3	User ID, Password, and New Password Authentication .....	518
10-4	User ID and Password Authentication .....	519
10-5	User ID and Password Substitute Authentication .....	520
10-6	User ID and Encrypted Password Authentication.....	521
10-7	Encrypted User ID and Password Authentication.....	522
10-8	Encrypted User ID, Password, New Password Authentication.....	523
10-9	Example Kerberos-Based Flow using SSPI and GSS-API in DRDA.....	526
10-10	Encrypted User ID and Security-Sensitive Data.....	528
10-11	Intermediate Server Encrypted User ID and Security-Sensitive Data .....	530
10-12	Encrypted User ID, Password, and Security-Sensitive Data .....	532
10-13	Encrypted User ID, Password, New Password, and Security-Sensitive Data .....	534
10-14	Example of Plug-In-Based Flows.....	536
11-1	Summary of Required Subvectors and Subfields.....	546
11-2	Subfield X'85' for Failure Causes Codepoint X'F0A3' .....	550
11-3	Subfield X'85's for Actions Codepoint X'32D1' .....	551
11-4	Subfield X'85' for Actions Codepoint X'32A0'.....	551
11-5	Subfield X'85's for Failure Causes Codepoint X'F0C0' .....	563
11-6	Major Vector/Subvector/Subfield Construction .....	566
11-7	Alert Example for AGNPRMRM with Severity Code of 64 (Part 1).....	567
11-8	Alert Example for AGNPRMRM with Severity Code of 64 (Part 2).....	568
11-9	Alert Example for AGNPRMRM with Severity Code of 64 (Part 3).....	569
12-1	DRDA Initialization Flows with LU 6.2 Security (Part 1) .....	584
12-2	DRDA Initialization Flows with LU 6.2 Security (Part 2) .....	585
12-3	DRDA Initialization Flows with DCE Security (Part 1) .....	586
12-4	DRDA Initialization Flows with DCE Security (Part 2) .....	587

12-5	DRDA Initialization Flows with DCE Security (Part 3) .....	588
12-6	DRDA Bind Flows (Part 1).....	591
12-7	DRDA Bind Flows (Part 2).....	592
12-8	DRDA Bind Flows (Part 3).....	593
12-9	DRDA SQL Statement Execution Flows (Part 1).....	594
12-10	DRDA SQL Statement Execution Flows (Part 2).....	595
12-11	Actual Flow: Termination Flows on SYNC_LEVEL(NONE) Conversation.....	597
12-12	Actual Flow: Termination Flows on SYNC_LEVEL(SYNCPT) Conversation.....	598
12-13	Commit Flow for a SYNC_LEVEL(NONE) Conversation.....	599
12-14	Actual Flow: Commit Flow on a SYNC_LEVEL(SYNCPT) Conversation.....	600
12-15	Actual Flow: Backout Flow on a SYNC_LEVEL(SYNCPT) Conversation.....	601
13-1	TCP/IP Components.....	612
13-2	DRDA Initialization Flows on TCP/IP with DCE Security .....	615
13-3	DRDA Bind Flows on TCP/IP .....	617
13-4	DRDA SQL Statement Execution Flows on TCP/IP.....	618
13-5	DRDA Termination Flows on TCP/IP .....	619
13-6	DRDA Server Abnormal Termination Flows on TCP/IP .....	619
13-7	DRDA Commit Flows on TCP/IP .....	620
13-8	TCP/IP Distributed Unit of Work Commit Flow .....	621
13-9	TCP/IP Distributed Unit of Work Rollback Flow .....	622
B-1	Scrollable Cursors: Example with qyrstblk Not Specified .....	716
B-2	Scrollable Cursors: Example with qyrstblk Not Specified .....	717

**List of Tables**

1-1	DDM Modeling and Description Terms .....	39
1-2	DDM Terms of Interest to DRDA Implementers.....	40
1-3	DDM Command Objects Used by DRDA .....	41
1-4	DDM Reply Data Objects Used by DRDA .....	42
4-1	DDM Commands Used in DRDA Flows.....	75
4-2	Access by the Minimum MGRLVLLS Parameter of EXCSAT and EXCSATRD.....	91
4-3	Security Mechanism to SECMEC Value Mapping .....	97
4-4	Example of Global Transaction .....	236
4-5	Example of Local Transaction .....	237
5-1	Data Objects, Descriptors, and Contents for DRDA Commands .....	256
5-2	QRYDSC with Default Formats .....	275
5-3	OUTOVR with One NOCL Override Triplet .....	276
5-4	SQL Result Set Field Usage .....	296
5-5	DRDA SQL Data Area Field Usage (DDM Level 6 and Above) .....	298
5-6	DRDA SQL Data Area Field Usage (DDM Level 6 and Above) .....	307
5-7	SQLDIAGSTT Field Descriptions .....	321
5-8	SQLCNGRP Field Descriptions .....	325
5-9	SQLDCGRP Field Descriptions.....	328
5-10	SQLDCXGRP Field Descriptions.....	331
5-11	MDD References Used in Early Environmental Descriptors.....	387
5-12	MDD References for Early Group Data Units .....	389
5-13	MDD References for Early Row Descriptors .....	390

5-14	MDD References for Early Array Descriptors .....	390
5-15	MDD References Used in Late Environmental Descriptors .....	390
5-16	MDD References for Late Group Data Units .....	390
5-17	MDD References for Late Row Descriptors .....	390
5-18	MDD References for Late Array Descriptors .....	391
5-19	TYPDEFNAM and TYPDEFOVR .....	392
5-20	Complete Set of Late Environmental Descriptors for QTDSQL370 ....	397
5-21	Complete Set of Early Data Unit Group Descriptors .....	399
5-22	Complete Set of Late Data Unit Descriptors .....	401
5-23	STATS Sample Table .....	401
5-24	EXECUTE IMMEDIATE Command Data .....	402
5-25	EXECUTE IMMEDIATE Reply Data .....	402
5-26	Open Query Command Data .....	403
5-27	Open Query Reply Data .....	404
5-28	Input Variable Array Command Data .....	406
5-29	Object Data Stream Example for Execution of CALL Statement .....	407
5-30	Reply Data Stream Example for Execution of CALL Statement .....	408
5-31	Reply Data Stream Example for Summary Component of Response .....	409
5-32	STATS Sample Table (Expanded) .....	411
5-33	Open Query Reply Data .....	412
7-1	Compatible SECMECs .....	459
7-2	Maximal Example for EXCSQLSTT .....	478
7-3	Maximal Example for EXCSQLSTT .....	479
7-4	Application Server Rules for OPNQRY, CNTQRY, CLSQRY, EXCSQLSTT .....	492
7-5	Application Requester Rules for FETCH .....	497
7-6	Application Requester Rules for CLOSE .....	500
8-1	DRDA Reply Messages (RMs) and Corresponding SQLSTATEs .....	503
11-1	Alerts Required for DDM Reply Messages .....	547
11-2	Additional Alerts Required at Application Requester .....	548
11-3	Alert Model AGNPRM .....	549
11-4	Alert Model BLKERR .....	553
11-5	Alert Model CHNVIO .....	554
11-6	Alert Model CMDCHK .....	555
11-7	Alert Model CMDVLT .....	556
11-8	Alert Model DSCERR .....	557
11-9	Alert Model GENERR .....	558
11-10	Alert Model PRCCNV .....	559
11-11	Alert Model QRYERR .....	560
11-12	Alert Model RDBERR .....	561
11-13	Alert Model RSCLMT .....	562
11-14	Alert Model SECVIOL .....	564
11-15	Alert Model SYNTAX .....	565
A-1	DDM Manager Relationship to DRDA Level .....	629
A-2	ABNUOWRM Reply Message Instance Variables .....	631
A-3	ACCRDB Command Instance Variables .....	632
A-4	Reply Objects for the ACCRDB Command .....	632
A-5	ACCRDBRM Reply Message Instance Variables .....	633
A-6	ACCSEC Command Instance Variables .....	634

A-7	Reply Objects for the ACCSEC Command .....	634
A-8	ACCSECRD Reply Object Instance Variables .....	635
A-9	AGNPRMRM Reply Message Instance Variables .....	636
A-10	BGNATMCHN Command Instance Variables .....	637
A-11	BGNBND Command Instance Variables .....	638
A-12	Command Objects for the BGNBND Command .....	638
A-13	Reply Objects for the BGNBND Command .....	639
A-14	BGNBNDRM Reply Message Instance Variables .....	640
A-15	BNDCPY Command Instance Variables .....	641
A-16	BNDDPLY Command Instance Variables .....	642
A-17	BNDSQLSTT Command Instance Variables .....	643
A-18	Command Objects for the BNDSQLSTT Command .....	643
A-19	Reply Objects for the BNDSQLSTT Command .....	643
A-20	CLSQRY Command Instance Variables .....	644
A-21	Reply Objects for the CLSQRY Command .....	644
A-22	CMDATHRM Reply Message Instance Variables .....	645
A-23	CMDCHKRM Reply Message Instance Variables .....	646
A-24	CMDNSPRM Reply Message Instance Variables .....	647
A-25	CMDVLTRM Reply Message Instance Variables .....	648
A-26	CMMRQSRM Reply Message Instance Variables .....	649
A-27	CNTQRY Command Instance Variables .....	650
A-28	Reply Objects for the CNTQRY Command .....	650
A-29	DRPPKG Command Instance Variables .....	651
A-30	Reply Objects for the DRPPKG Command .....	651
A-31	DSCINVRM Reply Message Instance Variables .....	652
A-32	DSCRDBTBL Command Instance Variables .....	653
A-33	Command Objects for the DSCRDBTBL Command .....	653
A-34	Reply Objects for the DSCRDBTBL Command .....	653
A-35	DSCSQLSTT Command Instance Variables .....	654
A-36	Reply Objects for the DSCSQLSTT Command .....	654
A-37	DTAMCHRM Reply Message Instance Variables .....	655
A-38	ENDATMCHN Command Instance Variables .....	656
A-39	ENDBND Command Instance Variables .....	657
A-40	Reply Objects for the ENDBND Command .....	657
A-41	ENDQRYRM Reply Message Instance Variables .....	658
A-42	ENDUOWRM Reply Message Instance Variables .....	659
A-43	EXCSAT Command Instance Variables .....	660
A-44	EXCSATRD Reply Object Instance Variables .....	661
A-45	EXCSQLIMM Command Instance Variables .....	662
A-46	Command Objects for the EXCSQLIMM Command .....	662
A-47	Reply Objects for the EXCSQLIMM Command .....	662
A-48	EXCSQLSET Command Instance Variables .....	663
A-49	Command Objects for the EXCSQLSET Command .....	663
A-50	Reply Objects for the EXCSQLSET Command .....	663
A-51	EXCSQLSTT Command Instance Variables .....	664
A-52	Command Objects for the EXCSQLSTT Command .....	664
A-53	Reply Objects for the EXCSQLSTT Command .....	665
A-54	GETNXTCHK Command Instance Variables .....	666
A-55	Reply Objects for the GETNXTCHK Command .....	666
A-56	INTRDBRQS Command Instance Variables .....	667
A-57	INTTKNRM Reply Message Instance Variables .....	668
A-58	MGRDEPRM Reply Message Instance Variables .....	669

A-59	MGRLVLRM Reply Message Instance Variables.....	670
A-60	OBJNSPRM Reply Message Instance Variables.....	671
A-61	OPNQFLRM Reply Message Instance Variables.....	672
A-62	OPNQRY Command Instance Variables .....	673
A-63	Command Objects for the OPNQRY Command .....	673
A-64	Reply Objects for the OPNQRY Command .....	674
A-65	OPNQRYRM Reply Message Instance Variables .....	675
A-66	PKGBNARM Reply Message Instance Variables .....	676
A-67	PKGBPARM Reply Message Instance Variables.....	677
A-68	PRCCNVRM Reply Message Instance Variables.....	678
A-69	PRMNSPRM Reply Message Instance Variables.....	679
A-70	PRPSQLSTT Command Instance Variables.....	680
A-71	Command Objects for the PRPSQLSTT Command .....	680
A-72	Reply Objects for the PRPSQLSTT Command .....	680
A-73	QRYNOPRM Reply Message Instance Variables .....	681
A-74	QRYPOPRM Reply Message Instance Variables .....	682
A-75	RDBACCRM Reply Message Instance Variables .....	683
A-76	RDBAFLRM Reply Message Instance Variables .....	684
A-77	RDBATHRM Reply Message Instance Variables.....	685
A-78	RDBCMM Command Instance Variables .....	686
A-79	Reply Objects for the RDBCMM Command .....	686
A-80	RDBNACRM Reply Message Instance Variables .....	687
A-81	RDBNFNRM Reply Message Instance Variables .....	688
A-82	RDBRLLBCK Command Instance Variables.....	689
A-83	Reply Objects for the RDBRLLBCK Command.....	689
A-84	RDBUPDRM Reply Message Instance Variables.....	690
A-85	REBIND Command Instance Variables .....	691
A-86	Command Objects for the REBIND Command.....	691
A-87	Reply Objects for the REBIND Command .....	691
A-88	RSCLMTRM Reply Message Instance Variables .....	692
A-89	RSLSETRM Reply Message Instance Variables .....	693
A-90	SECCHK Command Instance Variables .....	694
A-91	Command Objects for the SECCHK Command.....	694
A-92	Reply Objects for the SECCHK Command .....	694
A-93	SECCHKRM Reply Message Instance Variables .....	695
A-94	SNDPKT Command Instance Variables .....	696
A-95	Command Objects for the SNDPKT Command.....	696
A-96	Reply Objects for the SNDPKT Command .....	696
A-97	SQLERRRM Reply Message Instance Variables .....	697
A-98	SYNCCTL Command Instance Variables .....	698
A-99	Command Objects for SYNCCTL.....	698
A-100	SYNCCRD Reply Object Instance Variables .....	699
A-101	SYNCLOG Reply Object Instance Variables .....	700
A-102	SYNCRSY Command Instance Variables .....	701
A-103	Command Objects for SYNCRSY .....	701
A-104	SYNCRRD Reply Object Instance Variables.....	702
A-105	Reply Objects for SYNCRRD.....	702
A-106	SYNTAXRM Reply Message Instance Variables .....	703
A-107	TRGNSPRM Reply Message Instance Variables .....	704
A-108	VALNSPRM Reply Message Instance Variables.....	705



# Preface

## The Open Group

The Open Group is a vendor-neutral and technology-neutral consortium, whose vision of Boundaryless Information Flow™ will enable access to integrated information within and between enterprises based on open standards and global interoperability. The Open Group works with customers, suppliers, consortia, and other standards bodies. Its role is to capture, understand, and address current and emerging requirements, establish policies, and share best practices; to facilitate interoperability, develop consensus, and evolve and integrate specifications and Open Source technologies; to offer a comprehensive set of services to enhance the operational efficiency of consortia; and to operate the industry's premier certification service, including UNIX® certification.

Further information on The Open Group is available at [www.opengroup.org](http://www.opengroup.org).

The Open Group has over 15 years' experience in developing and operating certification programs and has extensive experience developing and facilitating industry adoption of test suites used to validate conformance to an open standard or specification.

More information is available at [www.opengroup.org/certification](http://www.opengroup.org/certification).

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 at [www.opengroup.org/bookstore](http://www.opengroup.org/bookstore).

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.

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published at [www.opengroup.org/corrigenda](http://www.opengroup.org/corrigenda).

## 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 referenced documents:

- Distributed Data Management (DDM)
- Formatted Data Object Content Architecture (FD:OCA)
- Structured Query Language (SQL) and Character Data Representation Architecture (CDRA)
- Distributed Transaction Processing (DTP)
- 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 Technical Standard, however, should be sent to the addresses provided on the copyright page.

# Trademarks

Boundaryless Information Flow™ and TOGAF™ are trademarks and Motif®, Making Standards Work®, OSF/1®, The Open Group®, UNIX®, and the “X” device are registered trademarks of The Open Group in the United States and other countries.

HP-UX® is a registered trademark of Hewlett-Packard Company.

The following are trademarks of the IBM Corporation in the United States and other countries:

AIX®  
AS/400®  
DATABASE 2®  
DB2®  
Distributed Relational Database Architecture®  
DRDA®  
IBM®  
MVS®  
Netview®  
OS/2®  
OS/390®  
OS/400®  
RISC System/6000®  
SQL/DS®  
System/390®  
VM®

Intel® is a registered trademark of Intel Corporation.

Microsoft® and Windows NT® are registered trademarks of Microsoft Corporation.

NFS® is a registered trademark and Network File System™ is a trademark of Sun Microsystems, Inc.

Solaris® is a registered trademark of Sun Microsystems, Inc.

VAX® is a registered trademark of Digital Equipment Corporation.

# *Referenced Documents*

These publications provide the background for understanding DRDA.

## **DRDA Overview**

For an overview of DRDA, read:

- DRDA, Version 4, Volume 1: Distributed Relational Database Architecture (DRDA), published by The Open Group (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:

- DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture, published by The Open Group.
- *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:

- Guide, February 1996, Distributed Transaction Processing: Reference Model, Version 3 (ISBN: 1-85912-170-5, G504), published by The Open Group.
- CAE Specification, November 1995, Distributed Transaction Processing: The CPI-C Specification, Version 2 (ISBN: 1-85912-135-7, C419), published by The Open Group.
- CAE Specification, February 1992, Distributed Transaction Processing: The XA Specification (ISBN: 1-872630-24-3, C193), published by The Open Group.
- Snapshot, July 1994, Distributed Transaction Processing: The XA+ Specification, Version 2 (ISBN: 1-85912-046-6, S423), 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. Comer, Prentice Hall, Englewood Cliffs, New Jersey, 1991, SC31-6144 (IBM).
- *Internetworking With TCP/IP Volume II: Implementation and Internals*, Douglas E. Comer, Prentice Hall, Englewood Cliffs, New Jersey, 1991, SC31-6145 (IBM).
- *Internetworking With TCP/IP*, Douglas E. Comer, SC09-1302 (IBM).
- *UNIX Network Programming*, W. Richard Stevens, Prentice Hall, Englewood Cliffs, New Jersey, 1990, SC31-7193 (IBM).

## Referenced Documents

- *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 OSF DCE security. A listing of security publications is available on The Open Group website at [www.opengroup.org](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.
- *The Open Group OSF DCE SIG Request For Comments 5.x*, GSS-API Extensions for DCE, available from The Open Group.
- *IETF RFC 1508*, Generic Security Service Application Program Interface, September 1993.
- *IETF RFC 1510*, The Kerberos Network Authentication Service (V5), September 1993.

The following publications contain useful information about security mechanisms:

- *FIPS PUB 81*, DES Modes of Operation (Cipher Block Chaining), December 1980, NIST.
- *FIPS PUB 180-1*, Secure Hash Standard, May 1993, NIST.
- *IETF RFC 1964*, The Kerberos Version 5 GSS-API Mechanism, June 1996.

The following publication contains useful information about applied cryptography:

- *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, Schneier, Bruce, published by Wiley, New York, c.1996, 2nd Edition.

### Data Definition and Exchange

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

- DRDA, Version 4, Volume 2: Formatted Data Object Content Architecture (FD:OCA), published by The Open Group.
- ISO/IEC 9075: 1999, Information Technology — Database Languages — SQL
- *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*.
- *IEEE DRAFT Standard for Floating Point Arithmetic*, P754; refer to <http://754r.ucbtest.org/drafts/754r.pdf>.
- *Densely Packed Decimal Encoding*, Cowlishaw M.F., IEE Proceedings — Computers and Digital Techniques, ISSN 1350-2380, Vol. 149, No. 3, pp102-104, May 2002.
- Technical Standard, October 1993, Application Response Measurement (ARM) Issue 4.0 - C Binding (ISBN: 1-931624-35-6, C036), published by The Open Group.
- Technical Standard, October 1993, Application Response Measurement (ARM) Issue 4.0 - Java Binding (ISBN: 1-931624-36-4, C037), published by The Open Group.
- World Wide Web Consortium (W3C); refer to [www.w4.org](http://www.w4.org).

# 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](#) 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)
- The Open Group Application Response Measurement (ARM)
- The Open Group XA+ Specification

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

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),
- DRDA application-directed Distributed Unit of Work (DUOW)/XA Distributed Transaction Processing (DTP)
- Initial support for DRDA database-directed Distributed Unit of Work (DUOW)
- DRDA database-directed Distributed Unit of Work (DUOW)/XA Distributed Transaction Processing (DTP)
- The totality of SQL-related functions

It is written with the intention of allowing 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 and database-directed Distributed Unit of Work/XA Distributed Transaction Processing.

## 1.1.2 What it Means to Implement DRDA Level 6

### XML Extensions

DRDA has been extended to support XML through the addition of new XML DRDA types at SQLAM Level 8.

- Support for new DRDA types XML String Internal Encoding and XML String External Encoding

These new types, based on the FD:OCA Generalized String types, are treated as externalized data flown in EXTDTAs.

The term “XML String” in DRDA refers to XML data in its serialized form. There is no restriction placed on the type of XML data that may be transported by the DRDA XML String types. For complete details on the format of serialized XML data, refer to World Wide Web Consortium ([www.w3.org/TR/xslt-xquery-serialization](http://www.w3.org/TR/xslt-xquery-serialization)).

Unlike traditional character data where the CCSID associated with the data is externally specified, XML by definition contains the encoding information within the data (see Section 4.3.3 of [www.w3.org/TR/1998/REC-xml-19980210](http://www.w3.org/TR/1998/REC-xml-19980210)). To support both the traditional SQL encoding mechanism and properly formed XML data, DRDA provides two XML types: XML String Internal Encoding to support XML consisting of its own encoding, and XML String External Encoding for XML data requiring an external CCSID to be associated to provide the encoding information.

- Support for the new CCSIDXML used to specify the default CCSID associated with the DRDA type XML String External Encoding

### Dynamic Data Format

Dynamic Data Format allows an FD:OCA Generalized String data in an answer set to be returned in a representation that is determined by the server at the time when the data is retrieved based on its actual length. This provides the server with the ability not to flow such data as an EXTDTA object when it is inefficient or impractical to do so. This support can be enabled by the application requester by specifying *dyndtafmt* with the value TRUE (X'F1') on OPNQRY or EXCSQLSTT that invokes a stored procedure that returns one or more result sets. Each representation is called a *mode*, and the following modes are supported:

X'01' Data in QRYDTA.

X'02' Data in EXTDTA.

X'03' Data as a progressive reference.

The data returned through this mechanism is considered to be small for Mode '01', medium for Mode X'02' and large for Mode X'03'. The length boundaries for each representation may be specified by the application requester at OPNQRY. Each FD:OCA Generalized String column in the QRYDTA consists of an FD:OCA Generalized String header containing a mode byte and the length portion of the value, and the data portion of the value may be returned in QRYDTA or EXTDTA, or as a progressive reference depending on the mode. See Data Format (DF Rules) in [Section 7.8](#) for more details. The Dynamic Data Format only applies to output data returned in an answer set or result set, and not to SQLDTA or SQLDTARD data objects.

Traditionally, a LOB in an answer set is flown from the server in a format requested specifically by the application requester, either as a LOB value or a LOB locator. Using Dynamic Data Format, the server determines the most efficient format for flowing the particular LOB data when it is retrieved based on its actual size, provided that no OUTOVR is specified. With no OUTOVR specified, the server can flow small LOBs in Mode X'01', medium LOBs in Mode X'02'

and large LOBs in Mode X'03'.

In order to enhance the sequential retrieval of large data, a new DDM command GETNXTCHK (Get Next Chunk) is introduced along with the progressive reference. It allows the application requester to specify a desired chunk length with the progressive reference, and the server to manage the progression of the reference through the data and return the subsequent chunk of the data of the requested length. It provides an optimization over using the SQL SUBSTR statement with the SQL LOB locator to achieve the same purpose.

This support involves the following:

- SQLAM manager at Level 8.
- Modifying the definitions of FD:OCA Generalized Byte String and Generalized Character String to include an extra byte to represent the mode of the data before the signed binary integer for the length.
- Adding OUTOVRNON to the OUTOVRROPT parameter. OUTOVRNON indicates that no output override will be specified on any of the CNTQRYs for the cursor; therefore, if the QRYPRCTYP is LMTBLKPRC, query data and all of its associated externalized data is returned in response to OPNQRY or EXCSQLSTT that invokes a stored procedure that returns one or more result sets, even if LOB data is present in the answer set. OUTOVRNON is only supported if Dynamic Data Format is enabled.
- On the requester:
  - Sending *dyndtafmt* on OPNQRY or EXCSQLSTT
  - Sending *smldtasz* on OPNQRY or EXCSQLSTT with a value ranging from 0 to QRYBLKSZ
  - Sending *meddtasz* on OPNQRY or EXCSQLSTT with a value ranging from *smldtasz* to 2147483647
  - Sending the GETNXTCHK command
  - Receiving the SQLCARD and EXTDTA objects in response to GETNXTCHK
  - Receiving the modified FD:OCA representation for Generalized Byte String and Generalized Character String
- On the server:
  - Receiving *dyndtafmt* on OPNQRY or EXCSQLSTT
  - Receiving *smldtasz* on OPNQRY or EXCSQLSTT with a value ranging from 0 to QRYBLKSZ
  - Receiving *meddtasz* on OPNQRY or EXCSQLSTT with a value ranging from *smldtasz* to 2147483647
  - Receiving the GETNXTCHK command
  - Sending the SQLCARD and EXTDTA objects in response to GETNXTCHK
  - Sending the modified FD:OCA representation of Generalized Byte String and Generalized Character String

### Improve Package Management

A new bind COPY command (BNDCPY) is added to help with the management of packages by allowing a package on a server to be copied to other servers or within the same server. This functionality allows better management of packages across a wider community by not requiring the knowledge of the content of the package to be known by the requester issuing the bind request. Bind options can change when copying a package.

A new bind DEPLOY command (BNDDPLY) is added to help with the management of packages by allowing a package on a server to be deployed to other servers or within the same server. A deploy request is similar to a copy request. It is significantly different from a copy request in that the package at the server is deployed intact to the designated target without any changes to the bind options which were used to create the source package. The only options permitted in the request are the specification of the owner of the target object and the qualifier/schema to use for unqualified object references within the SQL of the package.

New options on the DRPPKG command are added to allow the wildcarding of package names to help with the management of packages by allowing multiple packages to be dropped by a single drop package request. This functionality allows the following:

- All versions of all packages of all collections that the requesting user is allowed to drop on a server
- All versions of all packages of a specific collection that the requesting user is allowed to drop on a server
- All versions of a specific package of a specific collection that the requesting user is allowed to drop on a server
- The packages of all collections which match a specific name and/or version that the requesting user is allowed to drop on a server
- A specific version of all packages of all collections that the requesting user is allowed to drop on a server

The following DDM support is required:

- SQLAM manager at level 8.
- Support for the new BNDDPLY command. This command is used to request the deployment of a package from the target server to another database server or to a different collection within the target server.
- Support for the new BNDCPY command. This command is used to request the copy of a package from the target server to another database server or to a different collection within the target server.
- Support for the new RDBCOLIDANY, PKGIDANY, and VRSNAMANY options on the DRPPKG command. These options represent the parts of the fully-qualified package specification which can be wildcarded such that multiple packages qualify to be dropped.
- BNDCPY, BNDDPLY, and the DRPPKG (which has wildcarding specification) commands will result in the server returning extended diagnostic information regardless of the setting of DIAGLVL established during ACCRDB processing.

### IPv6 128-Bit IP Addressing

The ability for the requester and server to communicate via 128-bit IPv6 addresses. A new TCP/IP manager (CMNTCPIP) Level 8 is introduced to allow a requester and server to negotiate their IPv6 level of support. TCP/IP manager Level 8 allows the IPADDR instance variable to carry 128-bit IPv6 addresses. Note that when CMNTCPIP 8 is not negotiated, DRDA partners should still be prepared to honor CMNTCPIP Level 5 and thus generate and parse 32-bit IPADDR instance variables. The CRRTKN is extended to allow an IPv6 address to be represented. A SYNCLOG, returned as a result of a SYNCCTL Request Log command, must now contain two IPADDRs, one containing the server's IPv6 address and, optionally, another containing the server's mapped IPv4 address.

### Miscellaneous Added DRDA Types

The following additional DRDA types are supported as of SQLAM Level 8. Support for a new DRDA type involves supporting two new early environment LIDs and corresponding FD:OCA types.

- Boolean — support for new DRDA types (LIDs X'BE' and X'BF') and new corresponding FD:OCA types (field types X'25' and X'A5'). A boolean is a 2-byte quantity. 0 evaluates to FALSE and any other value evaluates to TRUE.

When communicating with partners at SQLAM Level 7 or lower, booleans are converted into 2-byte integers, DRDA types I2 and NI2 (LIDs X'04' and X'05').

- Binary string (Fixed Binary) — support for new DRDA types (LIDs X'C0', and X'C1'). When communicating with partners at SQLAM Level 7 or lower, the compatible data type is Fixed Bytes, DRDA types FB and NFB (LIDs X'26' and X'27').
- Varying-length binary string (Variable Binary) — support for new DRDA types (LIDs X'C2', and X'C3'). When communicating with partners at SQLAM Level 7 or lower, the compatible data type is Variable Bytes, DRDA types VB and NVB (LIDs X'28' and X'29').
- Decimal floating point — support for new DRDA types (LIDs X'BA' and X'BB') and new FD:OCA types (field types X'42' and X'C2'). This type of data is independent of machine format. To distinguish decimal floating point data from other DRDA floating point numbers (BF16, BF8, BF4), the following terminology is adopted:
  - Floating point — refers to data in any of the floating point formats supported by DRDA.
  - Basic floating point — refers to DRDA types BF16, BF8, BF4, which depend on the machine type (TYPDEFNAM).
  - Decimal floating point — refers to DRDA type DFP and is independent of machine type (TYPDEFNAM).

When communicating with partners at SQLAM Level 7 or lower, the compatible data type is the 8-byte basic floating point type supported by the sender's TYPDEFNAM, DRDA types BF8 and NBF8 (LIDs X'0A' and X'0B').

### N-Flow Security Authentication

Most of the security services — for example, GSS-API authentication — require multiple negotiations of security credentials between the client and the server before access is granted to the resource. A new SECCHKCD is being introduced to allow N-Flow negotiation of security credentials (SECTKN) between the application requester and application server.

DDM support required for this enhancement is as follows:

- DDM SECMGR support at Level 7 on both application server and application requester
- Application Server: Flowing of a new SECCHKCD value on the SECCHKRM, labelled *continue*.
- Application Requester: Recognizing the new SECCHKCD value.

### Support for Unprotected Downstream Updates over a Protected Connection

When an application requester is connected to an application server over a connection protected by the CMNSYNCPT or SYNCPTMGR, there may be a need for the application server to communicate with another downstream server. The downstream server may be a DRDA database server or it may be a server that requires the use of a non-DRDA communication protocol. In either case, the downstream server may only support an unprotected connection from the application server. Until now, since the upstream connection is protected, unprotected downstream updates cannot be allowed by the application server because atomicity cannot be ensured. However, a new feature is introduced that will allow an application requester to indicate to the application server whether such updates are permitted via the new unprotected update allowed (*unpupdalw*) option on the ACCRDB command. The application requester will only allow such updates if it is capable of guaranteeing transaction integrity under these circumstances by either ensuring that no other databases are updated within the unit of work, or by otherwise rolling back the unit of work. If allowed to perform such updates, the application server must inform the application requester when such an update has occurred for the first time within a unit of work via the new unprotected update (*unpupd*) option on the RDBUPDRM reply message.

The following DDM support is required:

- SQLAM manager at Level 8
- The optional use of the UNPUPDALW instance variable on the ACCRDB command
- The optional use of the UNPUPD instance variable on the RDBUPDRM reply message

### Trusted Application Server

This change supports the database concept of a Trusted Context which allows an establishment of a trust relationship between an application server and a database server. Once established, this relationship can be used to provide special privileges and capabilities to an external entity accessing a database server. When the database is accessed, a series of trust attributes are evaluated by the RDB to determine whether a trusted context is defined for the application server. The relationship between a connection and a trusted context is established when the connection is first created and that relationship remains for the life of that connection. The source security manager can provide additional end-user security attributes — such as the original end-user ID or the end user's registry name — to enable the server to perform identity mapping if the source and target user ID are different or contained in different user registries.

The following DDM support is needed by the application server and database server.

The RDB must be at Level 8 or higher and the SECMGR must be at Level 8 or higher if sending

any of the new optional security check parameters.

- A new optional TRUST parameter on the ACCRDB command to indicate the application server is requesting a special trust relationship between an external entity such as a middleware server with the RDB.
- A new optional TRUST parameter on the ACCRDB reply message to indicate the database server has established a special trust relationship between an external entity such as a middleware server with the RDB. If not returned, the connection is established with no special privileges.
- A new security mechanism EUSRIDONL allowing an encrypted user ID only mechanism on the SECCHK command.
- A new optional USRREGNM data object on the SECCHK command to identify the source security manager's user registry name.
- A new optional USRORGID data object on the SECCHK command to identify the application's original end-user ID if it is different than the ID passed in the USRID parameter. This value is provided for auditing purposes only.
- A new optional USRSECTOK object on the SECCHK command to identify the application's original end-user ID security token. The object is used for auditing purposes and the content is implementation-defined.

Once established, a trusted connection allows for a unique set of interactions between a database server, an application server, and an external entity including:

- The ability to reuse a connection under a different user ID and possibly use different security mechanisms to authenticate each user using the connection with a different security mechanism. If allowed by the database server, the external entity may not have to manage or pass credentials when switching the connection to a different user ID. This eliminates the need to manage end-user passwords or credentials by the external entity. Only the user establishing the trusted connection may be required to be authenticated with credentials.
- The ability to establish a connection using the local user ID when the source and target user ID registries are different. The target security manager uses the source user ID, the source registry name, and an optional security token to authenticate the end user and map the source user ID to the target user ID to be used by the RDB.
- The ability for an RDB to provide additional privileges to the external entity not available outside a trusted connection. This can be accomplished by the RDB associating special privileges to the trusted context used to establish a trusted connection.

#### **Increased Modification Levels in Product Identifier**

To support a larger number of modifications per release, the *M* field of the product identifier (PPPVVRRM) has been changed to allow uppercase letters A-Z as well as 0-9. The letters A-Z should be used after modifications 0-9 have been used (e.g., 0,1, ...,9,A,B,C, ...,X,Y,Z).

The following DDM support is required:

- RDB manager at Level 8

### 1.1.3 What it Means to Implement DRDA Level 5

#### Monitoring

The ability for a requester to request the server to return how long it takes to process a request. The server elapsed time allows the client to analyze where the cost of performing the request is occurring: in the network or in the server. Monitoring data is not returned as a new reply message, but as additional data to the reply. If monitoring fails or a specific type of monitoring is not supported, the request does not fail. Support for DRDA Level 5 consists of the following:

- On the requester:

Request AGENT at Level 7 and specify the new MONITOR instance variable on an SQL command and process the new MONITORRD reply object. The optional MONITOR instance variable is added to the following commands: CLSQRY, CNTQRY, DSCPVL, DSCRDBTBL, DSCSQLSTT, EXCSQLIMM, EXCSQLSET, EXCSQLSTT, OPNQRY, and PRPSQLSTT. The MONITOR instance variable is used to request the server to return the server elapsed time.

- On the server:

Support for AGENT at Level 7 which indicates support for the new MONITOR instance variable and the MONITORRD object. Currently, DRDA defines only the ability to return the elapsed time. No network costs are included in the time. If the server does not support the monitor type requested, the optional MONITORRD is not required to be returned in the reply.

#### SQL Long Identifiers

The maximum length for certain SQL long identifiers can now be up to 255 characters in length. To minimize the data stream size because of the increase in length, a default value is allowed to be used for package names and consistency tokens. Refer to the Connection Usage (CU Rules) in [Section 7.6](#) on using default package names. This requires SQL Application Manager (SQLAM) Level 7 support as follows:

DFTRDBCOL	Maximum length has been increased from 18 to 255 with no change in format.
PKGID	The length is fixed at 18 (with right blank padding if necessary) if the package identifier is 18 characters long or less. This is the same as the format used prior to SQLAM Level 7. As of SQLAM Level 7, if the package identifier is longer than 18, then the PKGID will be of the same length up to a maximum of 255 without right blank padding.
PKGOWNID	Maximum length has been increased from 8 to 255 with no change in format.
RDBCOLID	The length is fixed at 18 (with right blank padding if necessary) if the collection identifier is 18 characters long or less. This is the same as the format used prior to SQLAM Level 7. As of SQLAM Level 7, if the collection identifier is longer than 18, then the RDBCOLID will be of the same length up to a maximum of 255 without right blank padding.
RDBNAM	The length fixed at 18 (with right blank padding if necessary) if the RDB name is 18 characters long or less. This is the same as the format used prior to SQLAM Level 7. As of SQLAM Level 7, if the RDB name is longer than 18, then the RDBNAM will be of the same length up to a maximum of 255 without right blank padding.

PKGNAME, PKGNAMECSN, PKGNAMECT

The length is no longer fixed and is based on the lengths of the RDBNAME, RDBCOLID, and PKGID contained therein. As of SQLAM Level 7, the PKGNAMECSN instance variable is optional. If not specified, the PKGNAME is required to identify the package section number. The package name and consistency token defaults to the last set of values specified on the connection.

### New Security Mechanisms

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

- The User ID Encryption and Password Encryption Security Mechanism (EUSRIDPWD) specifies a method to encrypt both the user ID and password. This mechanism authenticates the user like the user ID and password mechanism, but the user ID and password are encrypted and decrypted using the encryption algorithm based on the ENCALG parameter, with encryption key length based on the ENCKEYLEN parameter. Diffie-Hellman public-key distribution is used to generate the connection key and shared private key.
- The User ID Encryption, Password Encryption, and New Password Encryption Security Mechanism (EUSRIDNWPWD) specifies a method to encrypt the user ID, password, and new password during a change password sequence. The user ID and passwords are encrypted and decrypted using the encryption algorithm based on the ENCALG parameter, with encryption key length based on the ENCKEYLEN parameter. Diffie-Hellman public-key distribution is used to generate the connection key and shared private key.
- The Kerberos Security Mechanism (KERSEC) specifies a method of authenticating users by the use of an encrypted Kerberos Ticket, as opposed to more conventional methods, such as user ID and password, which allow security tokens to flow across the network in plain text.
- The Strong Password Substitution Security Mechanism (PWDSSB) specifies the use of a password substitute. The difference between this and the existing Password Substitution Security Mechanism (PWDSBS) implemented in DRDA Level 4 is the algorithm used to generate the password substitute.

Encryption 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).

New security mechanisms are added to provide security for security-sensitive data, in addition to providing security during user authentication. The security-sensitive DDM/FD:OCA objects that are encrypted are SQLDTA, SQLDTARD, SQLSTT, SQLDARD, SQLATTR, SQLCINRD, SQLRSLRD, SQLSTTVRB, QRYDTA, EXTDTA, and SECTKNOVR.

- The User ID Encryption and Security-sensitive Data Encryption Security Mechanism (EUSRIDDTA) specifies a method to encrypt the user ID and security-sensitive data. Diffie-Hellman public-key distribution algorithm is used to generate the connection key and shared private key. The user ID and security-sensitive data are encrypted and decrypted using the encryption algorithm based on the ENCALG parameter, with encryption key length based on the ENCKEYLEN parameter.

- The User ID Encryption, Password Encryption, and Security-sensitive Data Encryption Security Mechanism (EUSRPWDDTA) specifies a method to encrypt the user ID, password, and security-sensitive data. Diffie-Hellman public-key distribution algorithm is used to generate the connection key and shared private key. The user ID, password, and security-sensitive data are encrypted and decrypted using the encryption algorithm based on the ENCALG parameter, with the encryption key length value based on the ENCKEYLEN parameter.
- The User ID Encryption, Password Encryption, New Password Encryption, and Security-sensitive Data Encryption Security Mechanism (EUSRNPWDDTA) specifies a method to encrypt the user ID, password, new password, and security-sensitive data. Diffie-Hellman public-key distribution algorithm is used to generate the connection key and shared private key. The user ID, password, new password, and security-sensitive data are encrypted and decrypted using the encryption algorithm based on the ENCALG parameter, with encryption key length based on the ENCKEYLEN parameter.

Data encryption mechanisms require the following DDM support:

- Security Manager (SECMGR) at Level 7
- The instance variables ENCALG and ENCKEYLEN are optional for ACCSEC and ACCSECRD.

### **Plug-In Security Mechanism**

A security device may choose to use plug-in modules to perform authentication. This affords it the flexibility to take advantage of many different underlying security mechanisms, all from a common interface—for example, GSS-API—as well as the extensibility to use authentication mechanisms beyond the DRDA-defined security mechanisms. This enhancement introduces support for a plug-in security mechanism and also allows for plug-in negotiation when multiple plug-in modules are present.

The following DDM support is required:

- Security Manager (SECMGR) at Level 7
- New PLGIN security mechanism
- The new data object PLGINNM which is optional for ACCSEC, but required for SECCHK if the SECMEC is PLGIN
- The new optional data object PLGINID for ACCSEC and SECCHK
- The required return of a PLGINLST reply data object following the ACCSECRD reply data object

The PLGINLST object is an ordered list consisting of a PLGINCNT data object followed by one or more PLGINLSE list entry data objects. Each PLGINLSE object will contain a PLGINNM data object and, optionally, a PLGINPPL data object and/or a PLGINID data object.

### Scrollable Cursors

The support for scrollable cursors<sup>1</sup> is optional in DRDA. If supported, it consists of the following:

- On the application requester:
  - Sending *qryrowset* parameter on OPNQRY
  - Receiving *qryattscr* parameter on OPNQRYRM
  - Receiving *qryattsns* parameter on OPNQRYRM
  - Receiving *qyattupd* parameter on OPNQRYRM
  - Sending *qryscrorn* parameter on CNTQRY
  - Sending *qryrownbr* parameter on CNTQRY
  - Sending *qryrowsns* parameter on CNTQRY
  - Sending *qryblkrst* parameter on CNTQRY
  - Sending *qyrtnnda* parameter on CNTQRY
  - Sending *qryrowset* parameter on CNTQRY
  - Sending *qryrowset* on EXCSQLSTT for a CALL statement
- On the application server:
  - Receiving *qryrowset* parameter on OPNQRY
  - Sending *qryattscr* parameter on OPNQRYRM
  - Sending *qryattsns* parameter on OPNQRYRM
  - Sending *qyattupd* parameter on OPNQRYRM
  - Receiving *qryscrorn* parameter on CNTQRY
  - Receiving *qryrownbr* parameter on CNTQRY
  - Receiving *qryrowsns* parameter on CNTQRY
  - Receiving *qryblkrst* parameter on CNTQRY
  - Receiving *qyrtnnda* parameter on CNTQRY
  - Receiving *qryrowset* parameter on CNTQRY
  - Receiving *qryrowset* parameter on EXCSQLSTT for a CALL statement

### New Begin Bind Option

PRPSTTKP is added to the Begin Binding a Package to an RDB command to control when prepared statements are released by a target RDB.

- Support the new PRPSTTKP instance variable on the BGNBND command for SQLAM Level 7.

---

1. The support for scrollable cursors in DRDA Level 5 supersedes the scrollable cursor support defined in DRDA Level 2; see [Section 1.1.6](#) (on page 35).

## Extended Describe

Descriptive information can be returned from a server on a prepare, a describe, a query, an execute, or on an execute of a stored procedure. This information allows the requester to provide descriptive information to the application. Some applications require more descriptive information than what is currently returned, while other applications require less descriptive information. In order to satisfy these requirements, describe processing is enhanced to allow a requester to control the amount and the type of information returned.

- On the requester:
  - Support for SQLAM Level 7.
  - Extended describe information can be returned on the DSCSQLSTT, PRPSQLSTT, OPNQRY, and EXCSQLSTT commands. Three types of SQLDA groups can be requested: a light, standard, or extended type. Each type provides a different level of descriptive information. The light SQLDA provides minimal descriptive information. The standard SQLDA provides the same descriptive information as in previous versions. The extended SQLDA provides additional descriptive information required by certain types of APIs such as JDBC.

Extended describe uses the RSLSETFLG, RTNSQLDA, and TYPSQLDA instance variables to control the level of descriptive information provided by the target server. The TYPSQLDA and RSLSETFLG identify the type of SQLDARD to be returned. The TYPSQLDA instance variable is added to the PRPSQLSTT, OPNQRY, and EXCSQLSTT commands. The RTNSQLDA instance variable is used to request optional descriptive information. The RTNSQLDA instance variable is added to the OPNQRY and EXCSQLSTT commands. The SQLCINRD provides different levels of column descriptive information for result sets as enumerated on the EXCSQLSTT command. The RSLSETFLG instance variable is used to identify if and the type of SQLDA returned in the SQLCINRD.

The SQLCIROW and SQLCIGRP descriptors are no longer used. The SQLUDTGRP is added to the SQLVRBGRP.

- Support the receiving of the three levels of descriptive information described by the enhanced FD:OCA SQLDARD array.
- On the server:
  - Support SQLAM Level 7.
  - Support the optional RTNSQLDA and TYPSQLDA instance variables on the PRPSQLSTT, DSCSQLSTT, OPNQRY, and EXCSQLSTT commands. The SQLCIROW and SQLCIGRP descriptors are no longer used.
  - Support the sending of the three levels of descriptive information described by the enhanced SQLDARD FD:OCA array.

## Greater than 32,767 Byte SQL Statements

Existing early descriptor character fields are mapped to a Variable Character Mixed or a Variable Character SBCS which allow a maximum of 32,767 bytes. SQL Statements described by the SQL Statement Group use these character fields. To allow SQL Statements to extend beyond the 32K limit, SQL statements are changed to map to nullable Large Character Objects Mixed and nullable Large Character Objects SBCS to allow for very large SQL Statements.

- On the requester:
  - Support for SQLAM Level 7.
  - Support the sending of the new SQLSTT and SQLATTR object as described by the enhanced FD:OCA SQLSTTGRP group.
  - Support the receiving of the new SQLSTT and SQLATTR object.
- On the server:
  - Support SQLAM Level 7.
  - Support the receiving of the new SQLSTT and SQLATTR object as described by the enhanced FD:OCA SQLSTTGRP group.
  - Support the sending of the new SQLSTT and SQLATTR object.

### Longer SQL Identifiers

In previous levels of the architecture, SQL identifiers were mapped to a Variable Character Mixed and Variable Character SBCS in FD:OCA descriptors with a maximum length of 30 characters. To allow SQL Identifiers to extend beyond the 30-character limit, the maximum length allowed is increased to 255 characters.

- On the requester:
  - Support for SQLAM Level 7.
  - Support the sending and receiving of the new level FD:OCA descriptors that increase the allowable size for SQL identifiers.
- On the server:
  - Support SQLAM Level 7.
  - Support the receiving and sending of the new level FD:OCA descriptors that increase the allowable size for SQL identifiers.

### Enhanced FD:OCA Local Identifiers (LID) Mapping

Every FD:OCA triplet (SDA, GDA, and RLO) is assigned a local identifier (LID). A LID is used as a short label to reference a triplet eliminating the need to flow the full descriptor. The previous level of the architecture allowed the assignment of up to 254 LIDs. With the addition of new data types, additional descriptors, and the use of override triplets, 254 unique values are not enough to support the growing number of triplets.

To increase the number of available LIDs, early and late descriptors are divided into two FD:OCA meta data reference types. Late descriptors continue to be described using the existing meta data reference type of X'01'. Early descriptors are described using a new meta data reference type of X'02'. Environmental descriptors are described in both meta data reference types allowing them to be used in both early or late descriptors. With this approach, early and late (SDA, GDA, and RLO) descriptors can share the same LID values and are distinguishable by the reference type of the DRDA meta data application class. A late override LID can only reference late descriptors and does block any references to an early descriptor. This frees up an additional 96 LIDs reserved for early descriptors and 24 LIDs reserved for environment descriptors.

- On the requester:

- Support for SQLAM Level 7.
- Since early descriptors are never generated or parsed by the requester, this change is expected to have minimal impact on existing implementations.
- On the server:
  - Support SQLAM Level 7.
  - This change is expected to have minimal impact to existing implementations.

### Interrupt Request

Interrupt request support allows an application requester to interrupt a DRDA request running at the application server. If both the application requester and the application server are at DRDA Level 5, RDB Level 7, and the application server supports interrupts, it will return the *rdbinttkn* parameter in the ACCRDBRM. An application requester at DRDA Level 5, RDB Level 7 must be able to accept a *rdbinttkn* returned in the ACCRDBRM.

### Multi-Row Input

Support for atomic multi-row insert which has been optional since DRDA Level 2 has been enhanced to include updates and deletes. In addition, support for non-atomic multi-row input has been added. Support for both flavors of multi-row input is optional. The operation is considered atomic if, in the event that any of the input data rows fails, then all other changes made to the database under this operation will be undone.<sup>2</sup> Otherwise, the operation is considered non-atomic.

The following DDM support is required:

- SQLAM manager at Level 7
- In the case of an SQL INSERT statement, the use of the BUFINSIND instance variable on the PRPSQLSTT command to indicate whether the buffered insert<sup>3</sup> technique should be used for an atomic multi-row input operation against a partitioned database
- The use of the ATMIND instance variable on the EXCSQLSTT command to indicate whether this is an atomic or non-atomic operation
- The use of the NBRROW instance variable on the EXCSQLSTT command to specify the number of rows
- The use of a null row to indicate to the server to skip a bad row for both atomic and non-atomic operations
- The return of one reply per input row when the operation is non-atomic

---

2. What changes get undone is server-specific. For example, all database changes should be backed out, but acquired locks may not be released.

3. Buffered insert is an optimization for enhancing the performance in inserting multiple rows into a table in a partitioned database by blocking the rows within the server.

### Atomic EXCSQLSTT Chaining

Support for the chaining of multiple EXCSQLSTTs as an atomic operation has been added. Being atomic means that if any of the EXCSQLSTTs within the chain results in an error, then all other changes made to the database stemming from any other EXCSQLSTT within the chain will be undone.<sup>4</sup> Support for this feature is optional.

The following DDM support is required:

- SQLAM manager at Level 7
- The use of the BGNATMCHN command to indicate the start of the atomic chain
- The use of the ENDATMCHN command to indicate the end of the atomic chain
- The use of the ENDCHNTYP instance variable on the ENDATMCHN command to indicate whether to abort the chain

### New Query Options

Options have been added for further customizing a query when a cursor is opened in the following aspects:

- Whether read locks should be released when the query is closed either implicitly by the server or explicitly by the application requester. The current behavior is that read locks are not released when the query is closed. This new option can be specified when a cursor is opened. As well, it can be specified when the application requester explicitly closes the query, in which case the setting will take precedence over what may have been specified previously for the query.
- Whether the server should close the query implicitly when there are no more rows (SQLSTATE 02000) for a non-scrollable cursor regardless of whether the cursor has the HOLD attribute specified or not. Currently, the server decides whether the query is closed implicitly or not without input from the application requester.

Also, for a query or stored procedure result set, the server may impose a limit of the number of rows that can be blocked at a time without influence from the application requester in terms of any parameter that may or may not have been explicitly specified on the command. This limit (called the blocking factor) may be based on a configuration setting on the server, or it may be the result of a clause on the SQL statement that is associated with the query. This value is now returned in the open reply in the OPNQRYRM reply message. The blocking factor is most useful to an intermediate server which is responsible for repackaging the query data to be sent to an application requester over DRDA or another communication protocol.

In addition, a database update can occur for a query either at OPEN time for a SELECT with INSERT statement, or when a row is fetched as a result of an associated UDF or trigger invocation. The server can now indicate to the requester that an update has taken place for a query in the reply to an OPNQRY, EXCSQLSTT, or CNTQRY command if necessary.

The following DDM support is required for all the enhancements above:

- SQLAM manager at Level 7
- The use of the QRYCLSRLS instance variable on the OPNQRY and CLSQRY commands to indicate whether read locks held by a cursor should be released when the query is closed either implicitly (upon SQLSTATE 02000 or otherwise a query terminating condition) or explicitly<sup>5</sup>

---

4. What changes get undone are server-specific. For example, all database changes should be backed out, but acquired locks may not be released.

- The use of the QRYCLSIMP instance variable on the OPNQRY command to indicate for a non-scrollable cursor whether the server should close the query implicitly when there are no more rows (SQLSTATE 02000)
- The return of the QRYBLKFCT instance variable on the OPNQRYRM reply message to indicate, where applicable, the blocking factor that is associated with the query
- The return of an RDBUPDRM reply message in the reply to the OPNQRY, EXCSQLSTT, or CNTQRY command to indicate a database update has occurred for a query subject to the Update Control (UP Rules) UP1 through UP4 (see [Section 7.19](#) (on page 467))

### Enhanced Bind Options

New bind option values for existing bind options have been added. Also, the BINDOPTVL (Bind Option Value) has been enhanced.

The following DDM support is required:

- SQLAM manager at Level 7
- Support for new enumerated values for the following bind options: BNDEXPOPT, DECPRC, STTDATFMT, and STTTIMFMT
- Support for BINDOPTVL values that are up to 32,767 bytes long

### Enhanced LOB Processing

When a LOB value is to be externalized in an EXTDTA reply data object, the sender (application requester or server) can now optionally defer indicating a nullable value is indeed null, and/or defer setting its length (in the FD:OCA Generalized String header), from the time its FD:OCA Generalized String header is generated within the FDODTA or QRYDTA reply data object, until the time the EXTDTA object itself gets generated.

The following DDM support is required for all the enhancements above:

- SQLAM manager at Level 7
- The sender (application requester or server) of an externalized LOB value can optionally turn on the high-order bit of its corresponding FD:OCA Generalized Strings header value in an FDODTA or QRYDTA reply data object to indicate that the length of the LOB value is unknown at the time the FD:OCA Generalized String header is generated.
- If the FD:OCA Generalized String header of an externalized non-nullable LOB value indicates an unknown length, its associated EXTDTA object can have a length of zero.
- In addition to having a null indicator associated with its FD:OCA Generalized String header value, a nullable externalized LOB value must also have a null indicator associated with its value in an EXTDTA. As always, if the null indicator for the FD:OCA Generalized String header indicates not null, there must be an EXTDTA object associated with it. However, the null indicator for the EXTDTA may then still indicate that the nullable LOB value is null. In order for the nullable LOB value not to be null, both null indicators have to indicate not null.

---

5. Note that not all read locks held by the cursor will necessarily be released; these locks may be held for other operations or activities.

## Streaming

Streaming has been introduced for QRYDTA and EXTDTA data objects for SQLAM manager Level 7. This mechanism allows the sender (application requester or server) to generate the relevant DDM object without explicitly setting its true length in the Layer B header. No DDM extended length field is ever employed when streaming is in use. While streaming is optional for the application requester or server as a sender, the receiver must be able to properly process a streamed data object. For details, refer to the DSS term in the DDM Reference.

## Simplified Query Processing Rules

Query processing rules are revised so that they are simpler to implement, can more readily allow extensions to the architecture, and yet allow the server greater flexibility in how to generate query reply data objects.

Blocking applies only to the QRYDTA reply objects.<sup>6</sup> Each query block is a QRYDTA DSS. The maximum query block size value allowed in the *qryblksz* parameter is increased from 32K to 10M, thus accommodating the larger data volumes required by modern, more data-intensive applications.

Two types of blocking behavior are supported, depending on the server's capabilities and preferences:

1. **Exact blocking:** The *qryblksz* is an exact value for a query block's size. Every query block, except for the last query block in the reply chain, must be exactly that size. If a row is larger than the query block size, then the row is blocked into QRYDTAs, each of which is exactly *qryblksz* in length, except for the last query block which may be shorter. If the query block can contain more than one row, then whole rows can be added to the query block until no more whole rows can be added (up to the number of rows allowed by the protocol in effect). If the query block is not the last query block in the reply chain, then the remaining space in the query block contains a partial row so that the total length is the exact query block size. If the query block is the last query block in the reply chain, then the server may either return a short query block (the query block size less the unused space) or may return a partial row in the remaining space.
2. **Flexible blocking:** The *qryblksz* specifies an initial value for a query block. Each QRYDTA contains the base row data for at least one complete row (in the case of single-row fetch) or the base row data for exactly one complete whole rowset (in the case of multi-row fetch). In the case of single-row fetch, if the QRYDTA can contain additional single rows, then the additional rows in the QRYDTA are also whole complete rows. If the space remaining in a QRYDTA can contain part of a row but not the whole complete row, then the QRYDTA is expanded beyond its initial size to contain the complete row. If this has occurred, then no additional rows may be added to the QRYDTA. Query block expansion can occur with any single row added to the query block, including the first row. In the case of multi-row fetch, only one rowset can be returned.

This supports involves the following:

- On the application requester:
  - Sending *qryblksz* on OPNQRY, CNTQRY, or EXCSQLSTT with a value of up to 10M
  - Receiving the *qryblktyp* parameter on OPNQRYRM

---

6. Restricting blocking to the QRYDTA eliminates the complexity of applying blocked behavior to other DSSs, such as OPNQRYRM, QRYDSC, and SQLCINRD, as was done in lower levels of the architecture.

- In response to OPNQRY, CNTQRY, or EXCSQLSTT, parsing query data returned in exact query blocks
- In response to OPNQRY, CNTQRY, or EXCSQLSTT, parsing query data returned in flexible query blocks
- In response to OPNQRY, CNTQRY, or EXCSQLSTT, parsing all other query reply DSSs as unblocked objects
- On the application server:
  - Receiving *qryblksz* on OPNQRY, CNTQRY, or EXCSQLSTT up to 10M
  - Sending the *qryblktyp* parameter on OPNQRYRM
  - In response to OPNQRY, CNTQRY, or EXCSQLSTT, either generating query data as exact query blocks or as flexible query blocks, depending on the server's capabilities or preferences
  - In response to OPNQRY, CNTQRY, or EXCSQLSTT, generating all other query reply DSSs as unblocked objects

### Command Source Identifier

In an environment where an application requester or database server multiplexes multiple sources within client applications or multiple client applications over a single database connection to a server, collisions can occur when multiple requests from more than one client application source try to operate against an identical section within a package. In such an environment, the package name, consistency token, and section number (*pkgnamcsn*) are no longer sufficient in identifying a unique invocation of a database command. Therefore, the command source identifier (*cmdsrcid*) is introduced to allow an application requester to uniquely identify the source of a database command operating on a particular section within a package in order to distinguish it from an identical command, albeit from a different source within the same application or from a different application altogether. For cursor operations, the command source identifier allows queries from multiple application sources to operate on a single section all within one package.

The application requester must ensure that all commands stemming from one application source take on the same command source identifier.

The following DDM support is required:

- SQLAM manager at Level 7
- The optional use of the CMDSRCID instance variable on the following commands: CLSQRY, CNTQRY, DSCSQLSTT, EXCSQLIMM, EXCSQLSTT, OPNQRY, and PRPSQLSTT.

### Query Instance Identifier

Prior to SQLAM Level 7, a query is uniquely identified by its package name, consistency token, and section number (*pkgnamcsn*). Still, from a single application source, it was not possible to open the query again unless the previous query was closed. In order to solve this issue, the concept of a query instance is introduced with this level of DRDA. This support makes it possible to open a cursor more than once from an application or stored procedure within a single invocation thereof, which may or may not be the desired behavior. In this regard, the new duplicate query allowed (*dupqryok*) option on the OPNQRY command allows the application requester to choose between the old and new behaviors. Furthermore, with the introduction of the command source identifier (*cmdsrcid*), it is now possible to have more than one query, each originating from a different application source, that all operate on a single *pkgnamcsn* within a

package. Therefore, the *pkgnamcsn*, the *cmdsrcid*, and the query instance identifier are required in order to uniquely identify the cursor opened by a particular open cursor invocation. For details on this feature, see [Section 4.4.6](#) (on page 145).

The following DDM support is required:

- SQLAM manager at Level 7
- The use of the DUPQRYOK instance variable on the OPNQRY command
- The return of the QRYINSID instance variable on the OPNQRYRM reply message
- The use of the QRYINSID instance variable on the CNTQRY and CLSQRY commands
- The use of the QRYINSID instance variable on the DSCSQLSTT command for any statement that has an open cursor associated with it
- The use of the QRYINSID instance variable on the EXCSQLSTT and EXCSQLIMM commands for a positioned DELETE/UPDATE SQL statement

### ARM Correlators

The Open Group Application Response Measurement (ARM) API is designed to measure the response time and status of transactions executed by application software. An ARM correlator stores contextual information associated with each work request (as defined in an ARM environment). Servers will accept ARM correlators on ACCRDB, and use this correlator to associate contextual information with each request. See ARM 4.0 for more information.

The support for ARM correlators requires SQLAM Level 7 and consists of the following:

- On the requester:
  - Support for the *armcorr* parameter on ACCRDB (see [Section 4.4.1](#) (on page 89))
- On the server:
  - Receipt of the *armcorr* parameter on ACCRDB and the semantics of receiving it (see [Section 4.4.1](#) (on page 89))

### Connection Health Check

The SNDPKT command is introduced to provide the ability to send and receive data packets of arbitrary size. The purpose of this DDM command is to test the connectivity by allowing between a requester and server. It can also be used to check whether the network connection is healthy. This command is similar to the *ping* utility described in [Chapter 11](#) (on page 539), which is used to determine the status of the network.

This command requires AGENT Level 7 and the following support:

- On the requester:
  - Sending the SNDPKT command
  - Receiving the PKTOBJ object in response to SNDPKT
- On the server:
  - Receiving the SNDPKT command
  - Sending the PKTOBJ object in response to SNDPKT

### Overriding Collection ID of Package Name

By grouping packages in a collection,<sup>7</sup> and creating multiple collections potentially containing identical package names, applications can switch between packages in different collections merely by indicating the desired collection, without other changes to application logic. For example, it may be desirable to create a test collection and a production collection, or to create multiple collections to allow different bind options to be in effect. A package path provides a list of collections in which a package may be found. In DRDA Level 5, a new package path specification mechanism is added to provide package switching functionality for SQL applications. This mechanism allows for overriding the collection ID of a package name that flows at execution time.

Prior to DRDA Level 5, the requester was responsible for resolving package references. The SET CURRENT PACKAGESET statement specifies a single collection in which to search for packages. The application could test for the existence of the package in the specific *packageset* (that is, collection), and repeat the process (setting CURRENT PACKAGESET to a different collection and testing for the existence of the package in that collection) as many times as necessary. The resolution of the package name was performed at the requester. When a list of collections is involved, this algorithm has drawbacks in terms of performance and inconvenience to the application, which must repeat the steps to search all the possible collections until the desired package is found. With DRDA Level 5, the CURRENT PACKAGE PATH value can be used to specify a list of qualifiers for packages which is used by the server during package resolution. This new mechanism results in reduced network traffic and an improvement in CPU/elapsed time for applications since it requires crossing the network only once to resolve the package name at the server, instead of crossing once per collection to perform resolution at the requester. The CURRENT PACKAGE PATH value is updated through a new type of environmental SET statement. Refer to [Section 7.17](#) for details on this new SET statement.

### New Reply Flows for DSCRDBTBL, DSCSQLSTT, and PRPSQLSTT

Currently for a DSCRDBTBL command, DSCSQLSTT command, or PRPSQLSTT command with RTNSQLDA set to true, in the case of an SQL error, as detected by the RDB, an SQLDARD reply data object must be returned. The architecture is enhanced to also allow an SQLCARD reply data object to be returned.

The following DDM support is required:

- SQLAM manager at Level 7
- If a DSCRDBTBL command, DSCSQLSTT command, or PRPSQLSTT command with RTNSQLDA set to true results in an SQL error, the server has the additional option of returning an SQLCARD reply data object which can optionally be preceded by an SQLERRRM reply message.

### XA Distributed Transaction Processing (DTP) Interface

An application that is accessing multiple resources within a distributed unit of work or transaction must ensure that data integrity is maintained at all resources. An application interfacing with an XA-compliant Transactional Manager (TM application) can achieve this by either using the services of the DRDA Sync point manager, which on behalf of the application will handle the task of protecting these resources. Or the TM application may decide that it wants to perform the task of protection. This support provides an XA Distributed Transaction Processing (DTP) interface that will enable an application requester to carry out the operations involved in protecting a DRDA resource, on behalf of the application.

---

7. In some environments, these package collections are known as *package schemas* (or simply *schemas*).

**Note:** DRDA does not describe the interface between the Application, XA-compliant Transactional Manager, and the Resource Manager. The DTP interface is not required in order to support the XA Manager (XAMGR).

The set of operations involved are:

SYNCCTL(New Unit of Work)

Register a transaction with the DBMS and associate the connection with the transaction's XID.

SYNCCTL(End Association)

End a transaction with the DBMS and dissociate the connection from the transaction's XID.

SYNCCTL(Prepare to Commit)

Request the application server to prepare a transaction for the commit phase.

SYNCCTL(Commit)

Commit the transaction.

SYNCCTL(Rollback)

Roll back the transaction.

SYNCCTL(Return Indoubt List)

Obtain a list of prepared and heuristically completed transactions at the application server.

SYNCCTL(Forget)

Ask the application server to forget about a heuristically completed transaction.

A new DRDA Manager, XA Manager (XAMGR), has been introduced to provide this transaction processing interface to the application. This TP interface has been modeled after the XA protocol defined in The Open Group XA+ Specification. The XAMGR is based on DDM *syncctl* and *syncprd* objects with the following enhancements required in implementing this support in the application requester and application server.

- On the application requester:
  - Request for XAMGR support
  - Flowing enhanced DDM *syncctl* objects

**Note:** These enhancements are only valid when using the XAMGR.

The enhancements include:

1. Addition of two new *synctype*s:
  - X'0B' End association with the Transactional Identifier (XID)
  - X'0C' Returning a list of prepared and heuristically completed XIDs
2. Addition of the XAFLAGS parameter on the *syncctl* request
3. Flowing of the RLSCONV when *synctype* X'0B' is specified
4. When using the services of the XAMGR, the UOWID, FORGET, and XIDSHR parameters cannot be specified on a SYNCCTL object.

- On the application server:
  - Support of XAMGR and the enhanced DDM sync point control flows
  - Flowing enhanced DDM *syncprd* objects

Enhancements to the *syncprd* objects include:

1. Addition of the PRPHRCLST and XARETVAl parameters on the reply object
2. Flowing the RLSCONV parameter when responding to *rlsconv* on a *syncctl* request
3. When using the services of the XAMGR, the parameter SYNCTYPE cannot be specified on a SYNCCRD object

From now on all connections that use the TP interface will be termed as XAMGR *protected connections*. The XAMGR also provides Local and Global Transaction support. A Global Transaction is a unit of work that involves multiple DBMSs operating in support of this unit of work. The TM is responsible for coordinating the transaction between the DBMSs using two-phase protocols. A Local Transaction is a unit of work against a DBMS over an XAMGR protected connection, where commit coordination is provided by the requester and does not require the use of the two-phase protocol.

### Resource Sharing for Protected Connections

This support allows the application server or database server to share recoverable resources so as to prevent resource deadlocks from other SYNCPTMGR protected connections involved in the same unit of work. This involves sending two resource sharing identifiers on a SYNCCTL(New Unit of Work) request, the XIDSHR, and the XID, whose name and format are based exactly on the Global Transaction Identifier used and defined in The Open Group XA+ Specification (see *DTP: The XA+ Specification* or the DDM XID for the format of the identifier). The use of the two identifiers allows the application server or database server to determine which set of SYNCPTMGR protected connections can share resources.

The XIDSHR indicator supports two modes of sharing, *partial* and *complete*. For partial sharing, the application server or database server is required to prevent deadlocks from other SYNCPTMGR protected connections whose XIDs match exactly. For complete sharing, the application server or database server is required to prevent deadlocks from other protected connections whose *Gtrid* part of the XID match; the *Bqual* is ignored (see the *DTP: The XA+ Specification* for the XID format). The UOWID still identifies the unit of work and not the XID. See the DDM Reference, SYNCPTOV for details. If the application requires that the transaction be identified by the XID, then use the XAMGR, not the SYNCPTMGR. Support for SYNCPTMGR Level 7 consists of the following:

- On the application requester:
  - Request the SYNCPTMGR at Level 7.
  - For SYNCPTMGR protected connections, flow the identifier (DDM XID instance variable) and the new indicator requesting the RDB to perform partial or complete sharing of RDB resources and locks (DDM XIDSHR instance variable) on the SYNCCTL(New Unit of Work) request.
  - For SNA protected conversations, flow the identifier (DDM XID) instance variable and the new indicator requesting the RDB to perform partial or complete sharing of RDB resources and locks (DDM XIDSHR instance variable) on the SYNCCTL(New Unit of Work) request prior to sending any SQL requests after an SNA syncpoint.
- On the application server:
  - Support the SYNCPTMGR at Level 7 which indicates that the application server or database server can share recoverable resources and locks across a set of protected connections that are identified by the XID parameter.
  - Support the processing of the transaction identifier (DDM XID) instance variable and a new instance variable to indicate to the RDB when to optimize shared resources and locks (DDM XIDSHR instance variable) on the SYNCCTL(New Unit of Work)

request.

### Reusing Network Connections

Users are concerned with the time it takes to connect to an application server. The creation of a connection for every application can be an expensive process, both from a system (CPU and memory) and a network (delays) standpoint. Users are also concerned about the number of physical connections to the application server when there are many concurrent client applications accessing an RDB due to the resources consumed for each connection. This is especially troublesome when most of the connections are dormant—that is, a user of an application may connect to an RDB and then rarely access data. In an effort to reduce these concerns, two types of connection reuse are defined: *connection pooling* and *transaction pooling*.

- **Connection Pooling**

Allows an application requester to reuse an existing network connection for a different application once an application disconnects from the connection either by terminating or by releasing the connection.

An application requester or application server at AGENT manager Level 7 indicates the support of flowing the EXCSAT, ACCSEC, SECCHK, and ACCRDB commands after a successful commit or rollback to initialize a connection on behalf of a new application.

- **Transaction Pooling**

Allows an application requester to share a network connection with other applications. An application requester can switch the application associated with a connection after a completion of a transaction. Transactions are delimited by the completion of a commit or rollback. During the commit or rollback process for remote unit of work or distributed unit of work connections, an application server indicates whether the connection can be reused by another application. If reuse is allowed, a group of SQL SET statements may be provided to establish the application execution environment prior to the execution of the next transaction for the same application on possibly another connection.

For a requester:

- Support AGENT manager Level 7, SYNCPTMGR Level 7 or SQLAM level 7, and RDB manager Level 7 which indicates support for transaction pooling.
- To determine whether the connection can be reused by another application, the requester issues the SYNCCTL command (if using SYNCPTMGR Level 7) or the RDBCMM/RDBRLLBCK command (if using SQLAM Level 7 on a non-protected connection) with the RLSCONV parameter set to REUSE. The application requester processes the reply based on the RLSCONV parameter on the SYNCCRD reply and any SQLSTT objects.
- Issue the EXCSAT, ACCSEC, SECCHK, ACCRDB, and EXCSQLSET (as required) commands with any SQLSTT objects returned on the SYNCCRD or ENDUOWRM reply to establish the application execution environment prior to executing another transaction for the application if the connection was reused by another application.

For a server:

- Support AGENT manager Level 7 and RDB manager Level 7 which indicates support for transaction pooling. If using a protected connection, a SYNCPTMGR Level 7 is required. If using an unprotected connection, SQLAM Level 7 is required.

- Process the SYNCCTL, RDBCMM, or RDBRLLBCK command with the RLSCONV parameter set to REUSE. If the connection can be reused by another application, generate the SYNCCRD or ENDUOWRM reply setting the RLSCONV parameter to REUSE and generate the SQLSTT objects to re-establish the application environment prior to executing another transaction for the application. Otherwise, the server completes commit processing specifying the RLSCONV parameter set to FALSE.
- Process the EXCSAT, ACCSEC, SECCHK, ACCRDB, and EXCSQLSET (as required) commands to establish the application environment when a connection is reused for a different application.

### Support for Input Variable Arrays

Support for input variable arrays is added to the SQLDTA. The SQLDTA is used to pass input data to the RDB. It flows as command data with the EXCSQLSTT command and the OPNQRY command. Input variable array data is identified by two additional FD:OCA data objects. These new data objects are required if the FDOTA contains a repeatable field. A repeatable field is used to describe an input array where each element in the array has the identical format, all having the same field length, field type, and type parameter. The existing FDODSC object describes all input, update, or parameter data for a single SQL statement. The descriptor carries type information by SDA references with zero extents for each input variable. The SDA extents for each field are implicitly provided prior to the FDODSC descriptor in the new FDOEXT data object. It contains the number of times each field is described in the FDODSC and repeated in the FDOTA. The FDOEXT data is described by the SQLNUMEXT early descriptor. After the FDOTA data object, offset values are provided for each field in the new FDOOFF data object. An offset value contains the relative offset in bytes to the start of the data from the start of the FDOTA. It is described by the SQLNUMOFF early descriptor. If the input data does not contain any repeatable fields, then the FDOEXT and the FDOOFF objects are not required.

Each input array represents multiple rows of a single column. An option on the array input statement identifies if the request is to succeed or fail as a unit, or if the database server is to proceed despite a partial (one or more rows) failure. The SQL clause to do this is ATOMIC or NOT ATOMIC where ATOMIC specifies that if the request for any row fails, then all changes made to the database by any other row including changes made by successful requests are undone. This is the default. When NOT ATOMIC is specified, the rows are processed independently. This means that if one or more errors occurs during the execution of the request, processing continues and any changes made during the execution of the statement are not backed out.

The following DDM support is required:

- On the requester:
  - SQLAM manager at Level 7
  - Support for the optional FD:OCA FDOEXT and FDOOFF data objects
  - Sending an SQLDTA with FDOEXT and FDOOFF data on the EXCSQLSTT and/or OPNQRY command
  - Receiving the SQLDIAGGRP in the SQLCARD indicating the status information for each failure or warning that occurred processing each database row
- On the server:
  - SQLAM manager at Level 7

- Support for the optional FD:OCA FDOEXT and FDOOFF data objects
- Receiving an SQLDTA with FDOEXT and FDOOFF data on the EXCSQLSTT and/or OPNQRY commands
- Sending the SQLDIAGGRP in the SQLCARD indicating the status information for each failure or warning that occurred processing each database row

### Enhanced Kerberos Security Mechanism

The Kerberos principal<sup>8</sup> for the server, which is required for generating the encrypted Kerberos V5 ticket by the application requester, must be predetermined at the server and then provided to the application requester manually prior to attempting the database connection using the Kerberos Security Mechanism (KERSEC). Because this is not always practical, an enhancement is introduced to allow the server to optionally return its Kerberos principal if and only if it claims KERSEC is one of its supported security mechanisms.

The following DDM support is required:

- SECMGR manager at Level 7
- The optional return of a KERSECPPL reply data object following the ACCSECRD reply data object

### SQLCARD upon Successful Connection

This enhancement allows the server to return an SQLCARD reply data object to the application requester for a successful database connection. The SQLCARD may contain an SQL warning and/or certain server-specific connect tokens. The SQL warning is usually of a nature that indicates to the user that corrective action should be taken after which the connection should be re-attempted, whereas connect tokens may include things like the partition number for a partitioned database.

The following DDM support is required:

- SQLAM manager at Level 7
- The optional return of an SQLCARD reply data object following the ACCRDBRM reply message

### Diagnostics Support

The current SQL Communication Area Reply Data was designed with a small number of fields for use by an RDB to provide error and warning information. The SQL Communication Area is used by the requester to determine the success or failure of the last SQL statement that was executed. These fields have, for the most part, been used up and the limitations imposed by the current SQL Communication Area structure are impacting the ability to obtain additional diagnostics about an error or warning from an RDB. For example, the SQL Communications Area SQL error message field is limited to only 70 characters. With the added support for large SQL identifiers, this field is not large enough to contain complete error tokens.

The contents of the SQL Communication Area are changed to include a new group that can be returned by the RDB. This new group is used to provide additional diagnostic fields as defined by the SQL GET DIAGNOSTICS statement. This includes information about the last statement

---

8. The Kerberos V5 principal is in the format:

`component / component / component@realm`

For details, refer to <http://search.ietf.org/internet-drafts/draft-ietf-krb-wg-kerberos-clarifications-00.txt>.

executed, such as the cursor attributes if the last statement allocated or opened an cursor. It includes additional warning or error conditions with a corresponding message optionally generated during the execution of the statement. For example, if the last statement generated an error or warning SQLCODE, a complete message text is provided. Also, connection information is returned indicating the connection attributes for each RDB that participated in the execution of the statement.

Requester support:

- Support for SQLAM Level 7.
- The requester must detect the SQL GET DIAGNOSTICS statement and not flow the statement to the server. It is a local-only statement. If the requester wants to receive the new diagnostic fields, DIAGLVL1 or DIAGLVL2 must be specified on the ACCRDB command. If the requester wants to receive the new diagnostic fields with no message text, DIAGLVL2 must be specified on the ACCRDB command.
- The requester generates connection diagnostics during connection processing unless the ACCRDBRM contains an SQLCARD with the connection diagnostic. The optional SQLCARD on the ACCRDBRM is new in this version.

Server support:

- Support for SQLAM Level 7.
- The server generates the SQLDIAGGRP if the ACCRDB command requests diagnostics; otherwise, the SQLDIAGGRP must be null. The SQLDIAGGRP group is optional for servers that do not support extended diagnostics even if requested on the ACCRDB command. If DIAGLVL1 is specified, the SQLDCMSG field should contain the message text for the condition. If DIAGLVL2 is specified, the SQLDCMSG message text fields should contain null strings.
- To minimize the amount of diagnostics returned for a query block, each row SQLCA is generated without connection or statement information. When generating the SQLCADTA in the query block, the SQLDIAGSTT group and the SQLDIAGCN array fields in the SQLCAGRP are set to null for each associated row data. An SQLCARD is generated after the initial query block if the ENDQRYRM and SQLCARD is not generated after the initial block. This SQLCARD that proceeds after the initial block provides the SQLDIAGSTT group and the SQLDIAGCN array diagnostic fields for the block.
- If an intermediate server is involved in executing the statement, each intermediate server appends an SQLDCNGRP to the end of the SQLDIAGCN array representing the connection from the intermediate server to the remote server.

### **Specify Statement Attributes when Preparing an SQL Statement**

Between the options that can be specified as part of the SELECT statement and the options that can be specified on DECLARE CURSOR, the combinations of these options are numerous. With the additional cursor attributes being introduced for cursor scrolling, the number of combinations is growing exponentially. With the existing architecture, dynamic SQL (for example, ODBC) applications would have to support a large number of DECLARE CURSOR statements to support all combinations. To minimize this constraint, attributes can now be specified when the SQL statement is prepared. Previously, all cursor attributes were hard-coded as part of the SQL statement itself. The attributes will continue to be supported as part of the statement for conformance with the SQL99 standard in support of static SQL, but now can be supported as part of prepare in support of dynamic SQL.

Statement attributes are to be sent as a new DDM data object with the PRPSQLSTT command.

The attributes character string is encoded using the SQLSTT FD:OCA descriptor and flows in a new SQLATTR DDM object. The optional SQLATTR object contains the attributes string for the statement that will be in effect if a corresponding attribute has not been specified as part of the associated SELECT statement. Leading and trailing blanks should be removed when generating the SQLATTR object.

The following DDM support is required:

- SQLAM manager at Level 7
- Generating and receiving the SQLATTR object on the PRPSQLSTT command

### Failover Support

The current Server List support is enhanced so that not only can it be used for load balancing on multi-homed servers, but it can also be used for providing connectivity information on alternate server locations where a database is replicated for failover. This information allows a requester to recover from a communications failure by connecting to the database at one such alternate server location once failover of the database is complete.

Also, in order to be able to reinstate the execution environment after reestablishing a connection to the database in the event of a communications failure, the requester can specify on the EXCSQLIMM, EXCSQLSET, EXCSQLSTT, or BGNATMCHN command that, should any special register setting be changed as a result of the command execution, for every special register whose setting has been modified on the current connection, the server must return an SQLSTT reply data object, which contains an SQL SET statement. With this knowledge, immediately after the database connection has been reestablished, either to the original server or an alternate failover server following a communications to restore the execution environment.

The following DDM support is required:

- SQLAM manager at Level 7
- For TCP/IP, specification of either a host name or an IP address in a server list entry
- The optional use of the RTNSETSTT instance variable on the EXCSQLIMM, EXCSQLSET, EXCSQLSTT, or BGNATMCHN command to indicate whether the server must return one or more SQLSTT reply data objects, each containing an SQL SET statement for any special register that has been modified on the current connection, if any special register setting has changed as a result of the command execution

### Rowset Cursors

A rowset cursor is a cursor defined such that more than one row can be returned for a single fetch statement called multi-row fetch. A multi-row fetch against a rowset cursor returns an SQL rowset. The requester is required to provide a statement-level SQLCA for the SQL rowset to the application.

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

- On the application requester:
  - Receiving *qryattset* on OPNQRYRM
  - Sending *qyrowset* on CNTQRY
- On the application server:
  - Sending *qryattset* on OPNQRYRM

— Receiving *qryrowset* on CNTQRY

### Intermediate Site Processing

A new object, MGRLVLOVR, allows a sending system to specify that a subset of the objects sent conform to the SQLAM manager level specified in the MGRLVLOVR rather than to the SQLAM manager level agreed upon during the EXCSAT/EXCSATRD exchange. The MGRLVLOVR object contains a manager level value that is less than the manager level agreed upon during the EXCSAT/EXCSATRD exchange. The intent of this support is to facilitate processing at an intermediate server by eliminating the need to perform character data, numeric data, or object format and content manipulations when passing objects between a source requester and a target server. This support involves the following:

- On the requester:
  - Receiving the MGRLVLOVR object and using the new SQLAM manager level to parse reply data objects
- On the server:
  - Sending the MGRLVLOVR object to change the SQLAM manager level that applies to reply data objects

For more information about intermediate server processing, refer to [Section 4.3.5](#) (on page 85).

## 1.1.4 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 7.17](#) 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.

The Password Encryption Security Mechanism (PWDENC) specifies a method to encrypt the password. This mechanism authenticates the user like the user ID 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 user ID are used as the DES encryption and decryption seeds.

The 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 247).

**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 Generalized String header 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.

### 1.1.5 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:

- Data Staging Area which is independent of the DRDA type of distribution.
- 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.
- 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.
- 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.
- Stored Procedures with result sets can be supported using a DRDA Distributed Unit of Work base.

- TCP/IP Communications manager can be supported on a DRDA RUOW base or DRDA DUOW base.

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.

### Data Staging Area

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

### 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 *pkgathrul* on BGNBND and the semantics of sending and processing it (see [Section 4.4.3](#) (on page 127))
- Support of *bndopt* object on BGNBND and the semantics of sending and processing generic bind options (see [Section 4.4.3](#) (on page 127))

### 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 user ID 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](#) and [Chapter 10](#) (on page 515))
- Security check (SECCHK) command and reply message (SECCHKRM) (see [Section 4.4.2](#) and [Chapter 10](#) (on page 515))
- SECVIOL alert (see [Table 11-1](#) and [Table 11-14](#) (on page 564))
- Support for at least one security mechanism outside of security provided by the network (see [Section 4.4.2](#) (on page 97))

### 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 70)) or TCP/IP Communications Manager (CMNTCPIP) at Level 5 (see [Section 4.3.1.3](#) (on page 71))
- 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.

### 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 127))

### 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 171))
- Handling commit and rollback in a stored procedure in a remote unit of work (see [Commit and Rollback Scenarios](#) (on page 219))
- Handling commit and rollback in a stored procedure in a distributed unit of work (see [Commit and Rollback Scenarios](#) (on page 219))
- Receipt of prcnamon EXCSQLSTT and the semantics of receiving it (see [Section 4.4.7.1](#) (on page 171))
- Result sets (see [Section 4.4.7.2](#) (on page 177))

### 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](#) and [Chapter 13](#) (on page 611))
- Security Manager (SECMGR) at Level 5

### 1.1.6 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<sup>9</sup>
- 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 4.4.15.2](#) (on page 217))
  - RDBUPDRM (see [Section 4.4.15.2](#) (on page 217))
  - CMDVLTRM (see [Section 4.4.15.2](#) (on page 217), [Table 11-1](#) (on page 547), and [Table 11-7](#) (on page 556))
  - Two-phase commit protocols (see [Section 3.1.4](#) (on page 61), [Commit and Rollback Scenarios](#) (on page 219), [Section 12.7.3.4](#) (on page 597), [Section 12.7.6](#) (on page 600), [Section 12.7.7](#) (on page 601), and [Section 12.7.9](#) (on page 602))
  - CRRTKN semantics and alert support (see [Section 11.3.2.2](#) (on page 544))
  - Coexistence rules (see [Section 4.4.15.2](#) (on page 217), [Section 12.7.8](#) (on page 602), and [Section 12.7.9](#) (on page 602))
  - CMDVLT alert (see [Table 11-1](#) and [Table 11-7](#) (on page 556))

---

9. This level of support for scrollable cursors is superseded by DRDA Level 5. See [Scrollable Cursors](#) (on page 12).

- On the application server:
  - CMMRQSRM (see [Section 4.4.15.2](#) (on page 217))
  - RDBUPDRM (see [Section 4.4.15.2](#) (on page 217))
  - CMDVLTRM (see [Section 4.4.15.2](#) (on page 217))
  - Two-phase commit protocols (see [Section 3.1.4](#) (on page 61), [Section 4.4.15.2](#) (on page 217), [Section 12.7.3.4](#) (on page 597), [Section 12.7.6](#) (on page 600), and [Section 12.7.7](#) (on page 601))
  - CRRTKN (semantics and alert support) (see [Section 11.3.2.2](#) (on page 544), [Table 11-1](#) (on page 547), and [Table 11-7](#) (on page 556))
  - CMDVLT alert (see [Table 11-1](#) and [Table 11-7](#) (on page 556))

### 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 247))
  - Support for QTDSQLASC (see [Chapter 5](#) (on page 247))
- On the application server:
  - Support for QTDSQLVAX (see [Chapter 5](#) (on page 247))
  - Support for QTDSQLASC (see [Chapter 5](#) (on page 247))

### Multi-Row Fetch

Support for this function at this level of DRDA was obsoleted in DRDA Level 5 and is removed from the DRDA Reference. Refer to [Rowset Cursors](#) for the support described by DRDA Level 5 (see [Section 1.1.3](#) (on page 9)).

### Multi-Row Insert

Support for this function at this level of DRDA was obsoleted in DRDA Level 5 and is removed from the DRDA Reference. Refer to [Support for Input Variable Arrays](#) and [Multi-Row Input](#) for the support described by DRDA Level 5 (see [Section 1.1.3](#) (on page 9)).

### Scrollable Cursors

Support for this function at this level of DRDA was obsoleted in DRDA Level 5 and is removed from the DRDA Reference. Refer to [Scrollable Cursors](#) for the support described by DRDA Level 5 (see [Section 1.1.3](#) (on page 9)).

### 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 the *dgriopr1* parameter on BGNBND (see [Section 4.4.3](#) (on page 127))

- Support for the *dgrioprl* parameter on REBIND (see [Section 4.4.5](#) (on page 143))
- On the application server:
  - Support for the *dgrioprl* parameter on BGNBND (see [Section 4.4.3](#) (on page 127))
  - Support for the *dgrioprl* parameter on REBIND (see [Section 4.4.5](#) (on page 143))

If not supported, the application server must still support:

- Receipt of *dgrioprl* on BGNBND and REBIND.

### CCSID Manager

The support for the 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](#) and [Section 4.4.1](#) (on page 89))
  - Support for DDM character command parameters in CCSIDs 819, 850, and 500; it might also support other CCSIDs (see [Section 4.3.1.13](#) and [Section 4.4.1](#) (on page 89))
- On the application server:
  - Support for specifying CCSIDMGR on EXCSATRD (see [Section 4.3.1.13](#) and [Section 4.4.1](#) (on page 89))
  - Support for DDM character command parameters in CCSIDs 819, 850, and 500; it might also support other CCSIDs (see [Section 4.3.1.13](#) and [Section 4.4.1](#) (on page 89))

## 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 The 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 39), it needs to be understood that every DRDA implementation needs to provide the DDM components and model structures listed in [Table 1-2](#) (on page 40). 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
AGENT	Agent
CMNAPPC	LU 6.2 conversational communications manager (introduced in DRDA Level 2)
CMNLYR	Communications layers
CMNMGR	DDM communications manager
CMNOVR	Communications overview
CMNSYNCPT	LU 6.2 sync point conversational communications manager (introduced in DRDA Level 2)
DCESECOVR	DCE security overview
DICTIONARY	Dictionary
DSS	Data Stream Structures
FDOCA	Formatted Data Object Content Architecture (FD:OCA)
MGROVR	Manager layer overview
OBJOVR	Object layer overview
RDB	Relational database
RDBOVR	Relational database overview
SECMGR	Security manager
SQL	Structured Query Language
SQLAM	SQL Application Manager
SQLDTA	SQL program variable data
SUPERVISOR	Supervisor
SYNCPTMGR	Sync point manager
XAMGR	XA Manager

### 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
<i>Connection establishment to a remote database manager</i>	
EXCSAT	Exchange server attributes
ACCRDB	Access RDB
<i>Package creation/rebind/remove</i>	
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
<i>Query Processing</i>	
OPNQRY	Open query
CNTQRY	Continue query
CLSQR	Close query
<i>Prepare/describe/execute SQL statements</i>	
PRPSQLSTT	Prepare SQL statement
DSCSQLSTT	Describe SQL statement
DSCRDBTBL	Describe RDB table
EXCSQLSTT	Execute SQL statement
EXCSQLIMM	Execute immediate SQL statement
<i>Commit/rollback unit of work</i>	
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
<i>Security processing</i>	
ACCSEC	Access security
SECCHK	Security check
<i>Propagating special register settings</i>	
EXCSQLSET	SET SQL environment
<i>Connection establishment to a remote database manager</i>	
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
ACCRDBRM	Access to RDB completed
ACCSECRD	Access security reply data
CMDCMPRM	Command processing completed
ENDQRYRM	End of query condition
ENDUOWRM	End unit of work condition
EXCSATRD	Server attributes reply data
OPNQRYRM	Open query complete
QRYDSC	Query answer set description
QRYDTA	Query answer set data
RDBUPDRM	Update at an RDB condition (Introduced in DRDA Level 2)
RSLSETRM	RDB result set reply message
SECCHKRM	Security check complete reply message
SECTKN	Security token reply data
SQLCARD	SQL communications area reply data
SQLCINRD	SQL result set column info reply data
SQLDARD	SQLDA reply data
SQLDTARD	SQL data reply data
SQLRSLRD	SQL result set reply data
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
SYNCLOG	Identifies the sync point log used for a unit of work
TYPDEFNAM	Data type definition name
TYPDEFOVR	Data type definition override

 *Technical Standard*

**Part 1:**

**Database Access Protocol**

*The Open Group*



# 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, The Open Group *Distributed Computing Environment (DCE)* for security support, and The Open Group *Distributed Transaction Processing (DTP) Reference Model* for transaction processing support.

For a better understanding of DRDA, some familiarity with these architectures is useful. See **Referenced Documents** 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 417).

**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](#) 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.<sup>10</sup> 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 four degrees of distribution of database management system functions. Each degree of distribution has different DRDA requirements. [Figure 2-1](#) illustrates the degrees of distribution.

---

10. 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.

<p>Application-Directed Remote Unit of Work (DRDA Level 1):</p> <ul style="list-style-type: none"> <li>• 1 DBMS per unit of work</li> <li>• Multiple requests per unit of work</li> <li>• 1 DBMS per request</li> <li>• Application initiates commit</li> <li>• Commitment at a single DBMS</li> </ul>
<p>Application-Directed Distributed Unit of Work (DRDA Level 2):</p> <ul style="list-style-type: none"> <li>• Several DBMSs per unit of work</li> <li>• Application directs the distribution of work</li> <li>• Multiple requests per unit of work</li> <li>• 1 DBMS per request</li> <li>• Application initiates commit</li> <li>• Commitment coordination across multiple DBMSs</li> <li>• 1 unit of work (uowid) per DBMS</li> </ul>
<p>Database-Directed Access (DRDA Level 4):</p> <ul style="list-style-type: none"> <li>• Several DBMSs per unit of work</li> <li>• Application directs requests to a DBMS</li> <li>• DBMS distributes the unit of work to multiple DBMSs</li> <li>• Multiple requests per unit of work</li> <li>• 1 DBMS per request</li> <li>• Application initiates commit</li> <li>• Commitment coordination across multiple DBMSs</li> <li>• Propagate special registers</li> </ul>
<p>XA Distributed Transaction Processing (DRDA Level 5):</p> <ul style="list-style-type: none"> <li>• Unit of work (transaction) identified by an XID</li> <li>• One application server per application requester instance</li> <li>• Multiple connections per application requester</li> <li>• Multiple SQL requests per XID</li> <li>• Application server can distribute SQL requests to other downstream database servers (via DSP)</li> <li>• Application requester, application server, and all database servers define the XA reply message</li> <li>• TM registers, coordinates, and recovers transactions with the registered application requester</li> <li>• Application server coordinates and recovers all downstream database servers involved in the XID</li> </ul>
<p>Application-Directed Distributed Unit of Work (DRDA Level 5):</p> <ul style="list-style-type: none"> <li>• DBMS uses XID to share recoverable resources between a set of protected connections.</li> </ul>

**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.<sup>11</sup> 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. With SYNCPTMGR Level 7 support, the sync point manager at the application server can share recoverable resources between a set of SYNCPTMGR protected connections when an XID is specified. The XID identifies to the sync point manager which protected connections are allowed to share recoverable resources, in order to prevent deadlocks. The XIDSHR allows the application to indicate how the RDB should identify this set of SYNCPTMGR protected connections.

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.

- *Application-Directed TP Access*

The XA-compliant TM is responsible for protecting any DBMS resources changed on the application server and any downstream database server. Before any work can be performed on a DBMS, the application requester must register the transaction with the DBMS and associate the connection with the transaction's XID using the SYNCCTL(New

---

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

Unit of Work) command. Before the application can begin commit coordination, it must end the transaction with the DBMS and dissociate the transaction's XID from the connection using the SYNCCTL(End Association) command.

Only one XID can be associated with a connection at any given time. But a transaction that has already been end'ed using the SYNCCTL(End association) command at a DBMS, and its XID is currently not associated or suspended on any connection, can be prepared, committed, or rolled back from any connection to that DBMS, even if that connection is associated with a different transaction. A recovery operation to the DBMS is valid at any given time, and will return a list of all XIDs that are currently in the prepared state or have been heuristically completed at the DBMS. The application server coordinates any XIDs with any downstream database servers associated with the application requester connection.

## 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 and application servers.

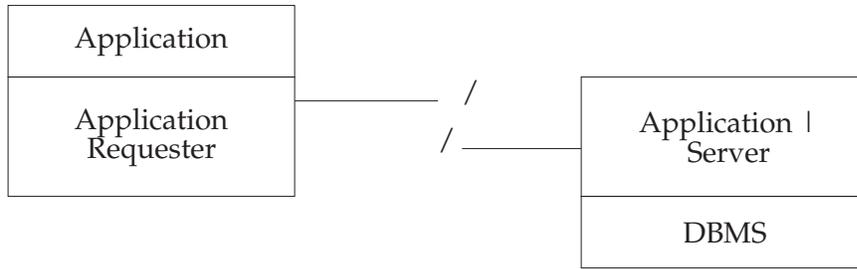
The application requester supports the application end of the DRDA connection by making requests to the application server, while the application server supports the database management system end by answering these requests.

- *Database Support Protocol*. Provides connections between application servers and database servers. 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 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](#) (on page 50).



Application Support Protocol

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 50).

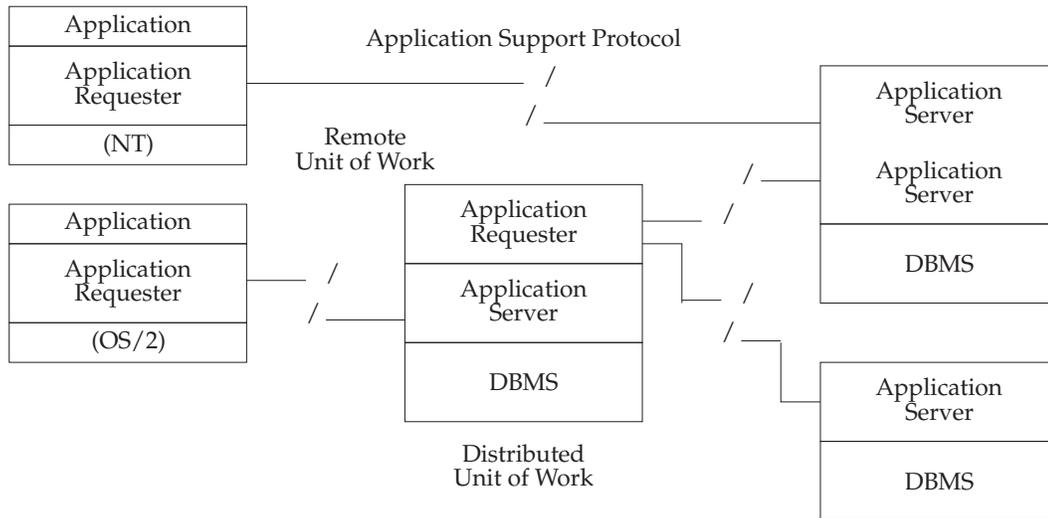


Figure 2-3 DRDA Network Implementation Example

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](#) for details on DDM managers.



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](#) for an explanation of the use of SNA verbs. Refer to [Chapter 4](#) for an explanation of the use of DDM commands and terms. Refer to [Chapter 5](#) 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 the plug-in security mechanism
- Through Distributed Computing Environment (DCE) security mechanisms
- Passing a user ID and password in a DDM command; optionally, these security tokens can be encrypted to prevent them from flowing in the clear
- Passing a user ID only (equivalent to already verified) in a DDM command
- Passing a user ID, password, and new password in a DDM command; optionally, these security tokens can be encrypted to prevent them from flowing in the clear
- 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](#) and [Figure 3-2](#) 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](#) assumes SNA security is used. [Figure 3-2](#) 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:

- User ID and password
- User ID only

- User ID, 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)

Here is the security mechanism ---->
I wish to use
(DDM ACCSEC)

I am USER_ID and here is
information to authenticate me
(DDM SECCHK)

I accept who you are and
I wish to access RDB_NAME
using DRDA flows with TYPDEF
(DDM ACCRDB)

I am RELEASE of AS from DRDA
ENVIRONMENT and support
the following subset of
requested DDM managers
(DDM EXCSATRD)

I accept your security
mechanism
(DDM ACCSECRD)

I accept who you are
and here is information to
authenticate me
(DDM SECCHKRM)

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](#) and [Figure 3-2](#) (on page 56), 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](#) and [Figure 12-3](#)
- For TCP/IP, [Figure 13-2](#)

### 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](#) 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](#) 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](#)
- For TCP/IP, [Figure 13-3](#)

### 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](#) 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<sup>12</sup>
- 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<sup>13</sup> and LMTBLKPRC in the DDM Reference for more details on fixed row and limited block protocols.

DRDA SQL statement execution flows for cursor operations OPEN, FETCH, and CLOSE also provide support for multiple instances of the same cursor which is a common occurrence in environments where User-Defined Functions (UDFs) and stored procedures returning result sets are used. For details on this feature, see [Section 4.4.6](#) (on page 145).

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

[Figure 3-4](#) 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.
- The query does not require application input variable values.

---

12. 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.

13. 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 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](#)
- For TCP/IP, [Figure 13-4](#)

### 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 or XA manager in use between the application requester and the 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 and XA manager use DDM-defined sync point control flows to commit the work. DDM flows are independent of the underlying communications manager.<sup>14</sup>

If an application requester requires Resource sharing between a set of protected connections at the application server or database server, it can send an XID instance variable on the DDM sync control new unit of work with an indicator specifying whether partial or complete sharing of RDB recoverable resources is required.

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 597).

[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.

---

14. 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.

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 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 599), or for TCP/IP refer to Figure 13-7 (on page 620).

### 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 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 SFNCCRD 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

### 3.1.6 Utility Flows

#### 3.1.6.1 Packet Flow

Packets will be flowed between application requester and application server. The purpose of packet flow is to test the connectivity between them. Furthermore, it can also be used to check whether the connection is healthy or not. The packet flow can occur at any time after the DDM EXCSAT command.

Figure 3-9 shows the type of information that flows between an application requester and application server.

Application End Information	DBMS End Information
I wish to check on my connection to RDBNAM by means of a response packet using PACKETSZ with request packet (DDM SNDPKT)	<--- Here is the response packet requested. (DDM PKTOBJ)

**Figure 3-9** Utility Flow: DRDA Packet Flows Using DDM Commands



## 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<sup>15</sup> 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

---

15. 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 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** 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 and the application server. The formats of these objects are described in [Chapter 5](#) (on page 247). 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 247). 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](#) and [Chapter 5](#) (on page 247). Additionally, the normal or usual usage of the DDM terms is described through the use of examples in [Section 4.4](#) (on page 88). For more background reading, see [DDM Guide](#) 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* (or, more simply, *requester*) in the DRDA processing model, and the DDM target system is referred to as the *application server* (or, more simply, *server*) in the DRDA processing model. DRDA also allows for more systems to be involved in the processing of a request, as described in [Section 2.5](#) (on page 49), but this section focuses on the behavior of the involved systems in a pairwise manner—generically, they are known as requester and server, even though the requester may be a database server sending the request to another database server. For additional discussion when more than two systems are involved, refer to [Section 4.3.5](#) (on page 85).

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](#) 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](#) (on page 70).

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, the application server, and the database server.

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 575).

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 575).

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 611).

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 515).

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 97). (For example, DCE security, user ID only, user ID 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.

- The Security Manager allows an application server to provide a local user ID on the security check command which can be different than the target user ID in the case when the source security manager and target security manager use different user registries. To allow the local user ID to map to the target user ID, a source user registry name is provided with the local or source user ID. If the source registry name provided is different than the target registry name used by the database server, the target security manager provides an association between the source and target registries. If an association is defined, the target security manager uses the source user ID, the source registry name, and the security token to authenticate the user and then map the source user ID to the target user ID. The target user ID is provided to the RDB when accessing the RDB.

The interactions between the source and target security managers are not defined, nor does the architecture resolve the problem of managing multiple user registries across different

systems and servers. It is left to the security manager implementers. There are several current industry approaches for solving the problem of managing multiple user registries; for example, LDAP provides a distributed user registry solution and EIM provides enterprise identity mapping across a group of systems and applications.

#### 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 application server).

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.

The sync point manager allows a set of protected connections at the application server or database server to share recoverable resources in order to prevent deadlocks, by specifying an XID informing the RDB which set of protected connections can share resources. An XIDSHR indicator allows the application to specify what degree of sharing is required.

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 the changed FD:OCA descriptors that the requester and server pass between them. FD:OCA descriptors defined in previous levels are no longer supported at this level. The sending or receiving of descriptors defined in previous versions of the architecture is a descriptor error (refer to [Section 5.7.3](#) and the DSCINVRM term in the DDM Reference for more details).

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 requester or application server is responsible for converting numeric and character data as necessary before sending data on the wire using the representative format it has chosen. An application requester or application server may select a representation which differs from its preferred format.

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.<sup>16</sup>

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 75). 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 codepoints) of the same names.

[DDM Guide](#) 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 7.11](#) for details on optional commands. [Section 4.4](#) discusses the allowed and recommended usage of these commands and some of the normal replies to these commands.

**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

16. Not supported in DRDA Remote Unit of Work.

DDM Command	Description
RDBRLLBCK	Rollback Transaction—rolls back (backs out) the current unit of work
EXCSQLSET (optional)	Propagate special registers
INTRDBRQS	Interrupt RDB Request—interrupts the DDM command currently executing.

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, state, 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.

To allow transaction pooling, an RDB provides an indicator to AGENT when an application server process can be reused by another application. If an application server process can be reused, the RDB provides a group of SQL SET statements, if necessary, to establish the current application environment on the next transaction for the application.

If the relational database manager supports interrupts, it may return a unique token identifier, interrupt token. If an interrupt token is returned, the relational database manager must be able to accept this token on another connection and interrupt the current DDM command executing on the original connection.

An RDB can be accessed by an application server as trusted to allow additional privileges and capabilities with an external entity, such as a middleware server, that is not available if not trusted. To create this trust relationship, a trusted context is required to be defined by the RDB to identify the entity's user ID and connection trust attributes. A connection trust attribute can be the client's IP address, a secure domain name that contains the client's IP address, required security mechanism, and required encryption level, such as requiring the use of SSL. The connection attributes are obtained from the communication manager or operating system using a set of secure interfaces. These connection attributes must match the attributes of a unique trusted context defined in order for the RDB to allow a trust relationship with the external entity. This RDB should implement an algorithm to determine whether a connection can be mapped to a trusted context. In order to minimize the chance in mapping connections to a trusted context, there needs to be a predictable, deterministic algorithm used for analyzing the connection attributes and evaluating the available trusted context definitions. The following logic can serve this purpose:

1. Search for any trusted context with the same entity's user ID used to establish the connection.
2. If a trusted context is found:
  - a. If the trusted context is disabled, this trusted context cannot be used.
  - b. If the trust attributes defined for this trusted context do not match the connection attributes used, this trusted context cannot be used. The connection is not trusted.

3. If a trusted context is not found and/or the connection is not trusted, then a warning is returned indicating the trusted connection could not be established for the entity's user ID. The connection is established, but without any extra privileges provided by a trusted context definition.

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.

#### 4.3.1.14 XA Manager

The XA manager is the system component responsible for providing a datastream architecture that will allow the application requester to perform the operations involved in protecting a resource. It provides the application requester with the following functionality:

1. SYNCCTL(New Unit of Work)
  - Registering a transaction with the DBMS and associating the connection with the transaction's XID.
2. SYNCCTL(End Association)
  - Ending a transaction with the DBMS and dissociating the connection from the transaction's XID.
3. SYNCCTL(Prepare to Commit)
  - Requesting the application server to prepare a transaction for the commit phase.
4. SYNCCTL(Commit)
  - Committing the transaction.
5. SYNCCTL(Rollback)
  - Rolling back the transaction.

6. SYNCCTL(Return Indoubt List)

Obtaining a list of prepared and heuristically completed transactions at the application server.

7. SYNCCTL(Forget)

Asking the application server to forget about a heuristically completed transaction.

The XAMGR on the application requester uses enhanced DDM SYNCCTL objects to convey the requests required to protect a resource to the application server. While the XAMGR on the application server conveys the response to the application requester using enhanced DDM SYNCCRD objects. The connection is always protected by a presumed rollback protocol in case of network failure or general errors.

### 4.3.2 The DRDA Processing Model Flow

Figure 4-1 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<sup>17</sup> in Figure 4-1 is only used if the local operating environment for the application requester and application server support sync point managers.

---

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

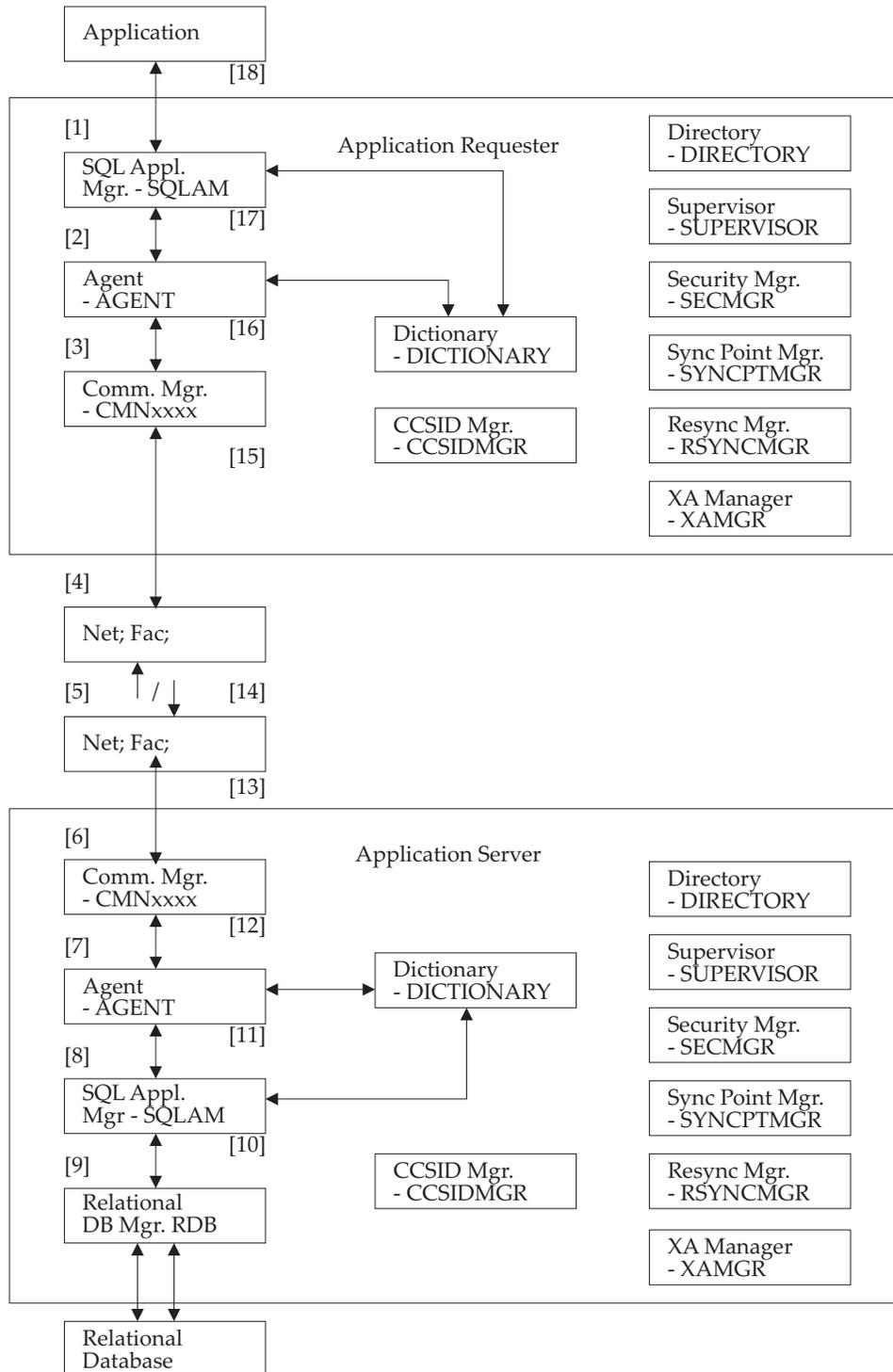


Figure 4-1 DRDA Processing Model

**Note:** The DDM Reference discusses in detail each of the managers represented in [Figure 4-1](#) (on page 79). 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 79).

DDM also groups managers into servers. In [Figure 4-1](#) (on page 79), 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](#) 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<sup>18</sup> or more DDM commands and associated parameters and command data.

The SQLAM uses functions modeled in the dictionary manager to determine the

---

18. 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.

codepoints (each DDM command, command parameter, data object, reply message, and reply message parameter has a unique codepoint) 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),<sup>19</sup> the character parameter values are sent in CCSID 500.

Command data objects are constructed using codepoints 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)<sup>20</sup> 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. If the command data object is to be encrypted, the communications manager accesses the security manager to encrypt the command data object, creates an encrypted object data stream structure (Encrypted OBJDSS), and places the encrypted command data object 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.

In any implementation of an application server, any manager that is modeled, must

---

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

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

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

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. If the data objects are encrypted, the communications manager accesses the security manager to decrypt the data objects. It 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 codepoints 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,<sup>21</sup> 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 and 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. Data representation conversion is not normally 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. An exception is when the target has specifically agreed to send data using a particular representation to reduce conversion overhead at the source.

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

---

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

(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. If the reply data object is encrypted, the communications manager accesses the security manager to encrypt the reply data object, creates an encrypted object data stream structure (Encrypted OBJDSS), and places the encrypted reply data object in it. Each OBJDSS 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 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. If the reply data objects are encrypted, the communications manager accesses the security manager to decrypt the reply data objects. It 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 89). 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 75), product-unique codepoints 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 codepoints used to represent characters, and the character data conversion of these codepoints, 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.

### 4.3.5 Intermediate Server Processing

#### 4.3.5.1 Overview and Terminology

In general, DRDA defines the interaction between two DRDA partners, one acting as the requester and one acting as the server. [Section 2.5](#) describes the terminology to be used when more than two systems are involved in processing a relational database management function. This section discusses the role of a server that receives a request for a relational database management function from another system but does not itself perform the function. To emphasize its role as being neither the system that originates the DRDA request nor the system that performs the relational database function, the database server is also known as an *intermediate server* for the request. In this capacity, the intermediate server acts as the server system to the requester (without distinction as to whether it is an application server or a database server). The requested function is sent by the intermediate server to another server. In this capacity, the intermediate server acts as the requester system to the other server. Scenarios in which an intermediate server are involved include gateway systems and multi-tier configurations, as described in [Section 2.5](#). Intermediate servers are also involved in the processing of some relational database management functions, such as the execution of a stored procedure on one server system where the stored procedure returns one or more result sets on another server system.

To facilitate discussion of this scenario, the following terminology applies in the case when one or more intermediate servers is involved in the processing of a relational database management request. The requester system that receives the request from the application is known as the *source requester*, and the server system that performs the requested function is known as the *target server*. Requests flow downstream from the application, through the intermediate servers, to the target server; replies flow upstream through the same path. See below.

Requests flow *downstream* from application to target server:

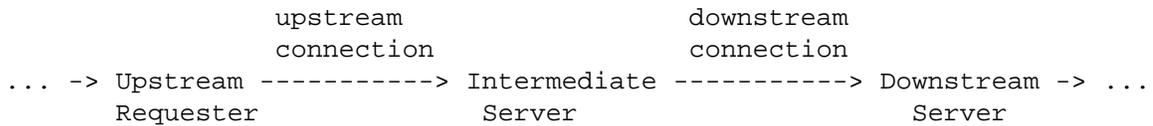
```
Application -> Source Requester -> ... -> Intermediate Server -> ... ->
Target Server
```

Replies flow *upstream* from target server to application:

```
Application <- Source Requester <- ... <- Intermediate Server <- ... <-
Target Server
```

From the perspective of an intermediate server, each intermediate server has at least two partners: an upstream requester and at least one downstream server. The upstream requester

may be the source requester or it may be another intermediate server. The downstream server may be the target server or it may be another intermediate server. The connection between the intermediate server and its upstream requester is known as the *upstream connection*, and the connection between the intermediate server and its downstream server is known as its *downstream connection*. See below.

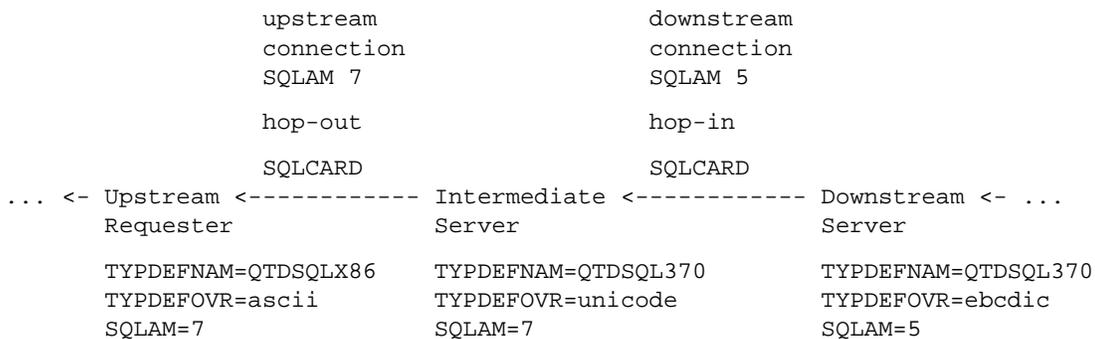


The DRDA processing model does not specify how an intermediate server behaves beyond describing its behavior as a server to the upstream requester and its behavior as a requester to the downstream server. In its capacity as intermediary between its two partners, the intermediate server must account for the differences that exist between the upstream connection and the downstream connection. For example, the upstream connection may have an associated SQLAM manager level or TYPDEFOVR or TYPDEFNAM that is different from what exists on the downstream connection. It is then the responsibility of the intermediate server to ensure that objects coming inbound on one connection are properly converted and reformatted according to the requirements of the other connection before they are sent outbound, either flowing upstream or downstream, on that connection.

Since the intermediate server does not itself perform the requested relational database management function, its primary function is one of transmitting properly-transformed-and-converted objects between its upstream requester and its downstream server. The process by which an intermediate server sends requests downstream or sends replies upstream is also known as *hopping*. The requests or replies are said to hop in from one partner and, after the proper conversion and transformation, to hop out to the other partner. When an intermediate server sends objects that hop out to its partner, the intermediate server is said to *hop* those objects.

#### 4.3.5.2 Examples

Below is a simple example of intermediate server processing. The example assumes that each server has a different CCSID, indicated in the diagram generically as *ebcdic*, *unicode*, or *ascii*. In the figure, the upstream requester has an EBCDIC CCSID (such as CCSID 37), the intermediate server has a Unicode CCSID (such as CCSID 1208), while the downstream server has an ASCII CCSID (such as CCSID 437). In this example, the downstream server has sent an SQLCARD object, as defined by SQLAM manager Level 5, with character and numeric data as specified by the downstream server's default TYPDEFOVR(*ebcdic*) and TYPDEFNAM(QTDSQL370). The reply object is hopped out by the intermediate server according to the requirements of SQLAM Level 6, with character and numeric data as specified by the intermediate server's default TYPDEFOVR(*unicode*) and TYPDEFNAM(QTDSQL370).



Alternatively, an intermediate server may hop objects, without transformation, by using one or more of the override objects, TYPDEFNAM, TYPDEFOVR, or MGRLVLOVR.<sup>22</sup> See below for an example. In this example, the downstream server has sent an SQLCARD object, as defined by SQLAM manager Level 5, with character and numeric data as specified by the downstream server's TYPDEFOVR(*ebcdic*) and TYPDEFNAM(QTDSQL370). The intermediate server prepends the reply chain with a MGRLVLOVR object and TYPDEFOVR object to override the intermediate server's default values. In this case, the intermediate server does not need to transform or convert the hop-in SQLCARD before hopping it out.

	upstream		downstream
	connection		connection
	SQLAM 7		SQLAM 5
	hop-out		hop-in
	MGRLVLOVR(SQLAM=5)		
	TYPDEFOVR( <i>ebcdic</i> )		
	SQLCARD		SQLCARD
...	<- Upstream <-----	Intermediate <-----	Downstream <- ...
	Requester	Server	Server
	TYPDEFNAM=QTDSQLX86	TYPDEFNAM=QTDSQL370	TYPDEFNAM=QTDSQL370
	TYPDEFOVR=ascii	TYPDEFOVR=unicode	TYPDEFOVR=ebcdic
	SQLAM=7	SQLAM=7	SQLAM=5

See [Section 7.9](#) for information about the use of TYPDEFNAM and TYPDEFOVR.

See [Section 7.6](#) for information about the use of MGRLVLOVR.

---

22. MGRLVLOVR is not supported in SQLAM Level 6 and below.

## 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 and an application server.<sup>23</sup> These flows are equivalent between an application server and a database server 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. Furthermore, the general term *requester* may be substituted whenever a system acts as an application requester sending requests to an application server, or as an application server sending requests to a database server, or as a database server sending requests to another database server. The general term *server* may also be substituted whenever a system acts as an application server receiving requests from an application requester or as a database server receiving requests from an application server or another database server. In particular, a requester or server may be an intermediate server, as described in [Section 4.3.5](#) (on page 85). Moreover, the MGRLVLOVR object may be sent as a reply data object in the flows, even though it is not explicitly listed.

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.

---

23. 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.

### 4.4.1 Accessing a Remote Relational Database Manager

Figure 4-2 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.

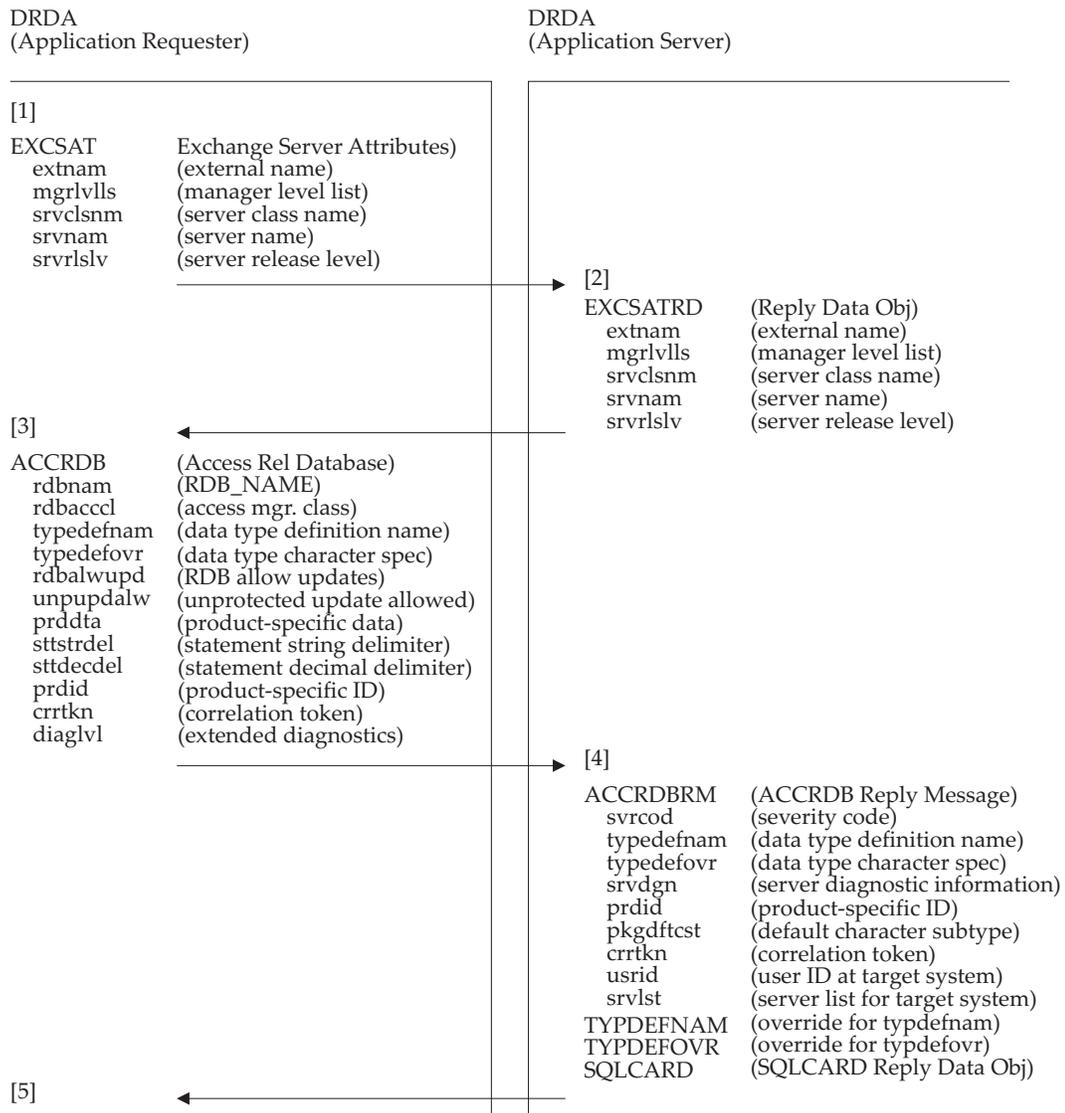


Figure 4-2 Establishing a Connection to a Remote Database Manager

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")
    mgrlvlsls(
        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 *mgrlvlsls* is the minimum list necessary to determine that the source and target manager levels are compatible for DRDA functions. The *mgrlvlsls* 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.

**Note:** At an intermediate server, additional manager-level control is obtained through the use of the MGRVLVLOVR object. See [Section 4.3.5](#) (on page 85).

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 *mgrlvl*s 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](#) (on page 91), summarizes the types of read and write access that can be accomplished based on the *mgrlvl*s specified on EXCSAT and EXCSATRD.<sup>24</sup>

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

AGENT	3	4	4	5	5	5
SQLAM	3	4	4	4	4	4
CMNSYNCPT			4			
SYNCPTMGR				5	7	
XAMGR						7
RUOW Access with Single-RDB Access	yes	no	no	no	no	no
DUOW Access with Multi-RDB Read and Single-RDB Write	no	no	yes	yes	yes	yes
DUOW Access with Multi-RDB Read and Multi-RDB Write	no	no	yes	yes	yes	yes
DUOW Access with Multi-RDB Read, Multi-RDB Write, and Resource Sharing	no	no	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 *srvcslnm* 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/dbiop/index.htm>.

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

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

- The *srvmam* 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.

- The *svrslsv* 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,<sup>25</sup> 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*, *srvmam*, or *svrslsv*.

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")
  mgrlvlsls(
    mgrlvl(AGENT,5)
    mgrlvl(SECMGR,5)
    mgrlvl(CMNTCPIP,5)
    mgrlvl(SYNCPTMGR,5)
    mgrlvl(SQLAM,5)
    mgrlvl(CCSIDMGR,500)
    mgrlvl(RDB,3))
  srvclsnm("QDB2")
  srvmam("STLDB2A1")
  svrslsv("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.

---

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

- The application server returns the *mgrlvl*s, 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 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.

**Note:** At an intermediate server, additional manager-level control is obtained through the use of the MGRVLVLOVR object. See [Section 4.3.5](#) (on page 85).

If the application requester cannot accept the CCSID *mgrlvl*<sup>26</sup> received from the application server, the conversation is terminated.

- The *srvcslnm*, *srvcnam*, and *srvcslv* 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*, *srvcslnm*, *srvcnam*, or *srvcslv* except those that are needed to determine if the application requester and application server are a specific product pair (see [Section 4.3.3](#) 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 *rdbaccl* 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 5-19](#) 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 5-19](#) for details.

---

26. The *mgrlvl*s 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 *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 the execution of a commit or rollback that the DDM commands EXCSQLIMM or EXCSQLSTT requested.

- The *unpupdalw* parameter specifies whether the application requester can enforce commit rules required for preserving transaction integrity over a CMNSYNCPT or SYNCPTMGR-protected connection, should the application server perform an update against another downstream DRDA or non-DRDA server over an unprotected connection. For details, refer to the CR Rules in [Section 7.5](#) (on page 432).

If the application requester indicates that it cannot ensure transaction integrity for this type of update, the application server is prohibited from performing such updates.

- The *prddta* parameter specifies product-specific information<sup>27</sup> that the application requester conveys to the application server if the *svclsnm* 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/dbiop/index.htm> 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 either a letter 0-9 or A-Z (0 means no modification level associated with the level of the product). Modifications increase through 0-9 followed by uppercase A-Z making Z the largest modification available (e.g., 0,1, ...,9,A,B,C, ..., X,Y,Z)).

- The *diaglvl* parameter can request standard or extended diagnostics if an SQL statement fails. If set to extended, the requester is requesting the server provide a non-null FD:OCA SQLDIAGGRP group with the FD:OCA SQLCAXGRP group. The SQLDIAGGRP group contains additional diagnostics information on why the SQL statement failed. If the parameter is not specified or the default value of standard is specified, the SQLDIAGGRP must be returned as a null group.

---

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

- 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 11.3.2.2](#) for details on setting the value of this token.
- The *armcorr* parameter indicates the ARM correlator to associate with all requests for this connection. This parameter is new in DRDA Level 5.

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](#) for details.

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](#) for details.

- The application server returns the *prdid* parameter, specifying the current release level of the application server.
- 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 or TCP/IP host names that can be used to access the RDB. The list can be used by the requester to work load balance future connections. It may also be used by the server to indicate one or more alternate failover locations where the database is replicated. This parameter is new in DRDA Level 3. Details of the server list and examples are in the DDM Reference.

The application server may optionally return an SQLCARD reply data object containing an SQL warning and/or server-specific connect tokens after the ACCRDBRM reply message.

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 an SQLCARD reply data object follows the ACCRDBRM reply message, the application requester may return the SQL warning and/or server-specific connect tokens contained therein to the application through the SQLCA.

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 97). (For example, DCE security, user ID only, user ID 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
Kerberos	<i>kersec</i>
Plug-in	<i>plgin</i>
DCE	<i>dcesec</i>
User ID only	<i>usridonl</i>
User ID and password	<i>usridpwd</i>
Encrypted user ID and password	<i>eusridpwd</i>
User ID and encrypted password	<i>usrencpwd</i>
User ID and password substitute	<i>usrsbpwd</i>
User ID, password, and new password	<i>usridnpwd</i>
User ID and strong password substitute	<i>usrssbpwd</i>
Encrypted user ID only	<i>eusridonl</i>
Encrypted user ID, password, and new password	<i>eusridnpwd</i>
Encrypted user ID and security-sensitive data	<i>eusriddda</i>
Encrypted user ID, password, and security-sensitive data	<i>eusripwdda</i>
Encrypted user ID, password, new password, and security-sensitive data	<i>eusripwdda</i>

More information on the security mechanisms is available in [Chapter 10](#) (on page 515).

Figure 4-3 shows the Kerberos or DCE security flows:

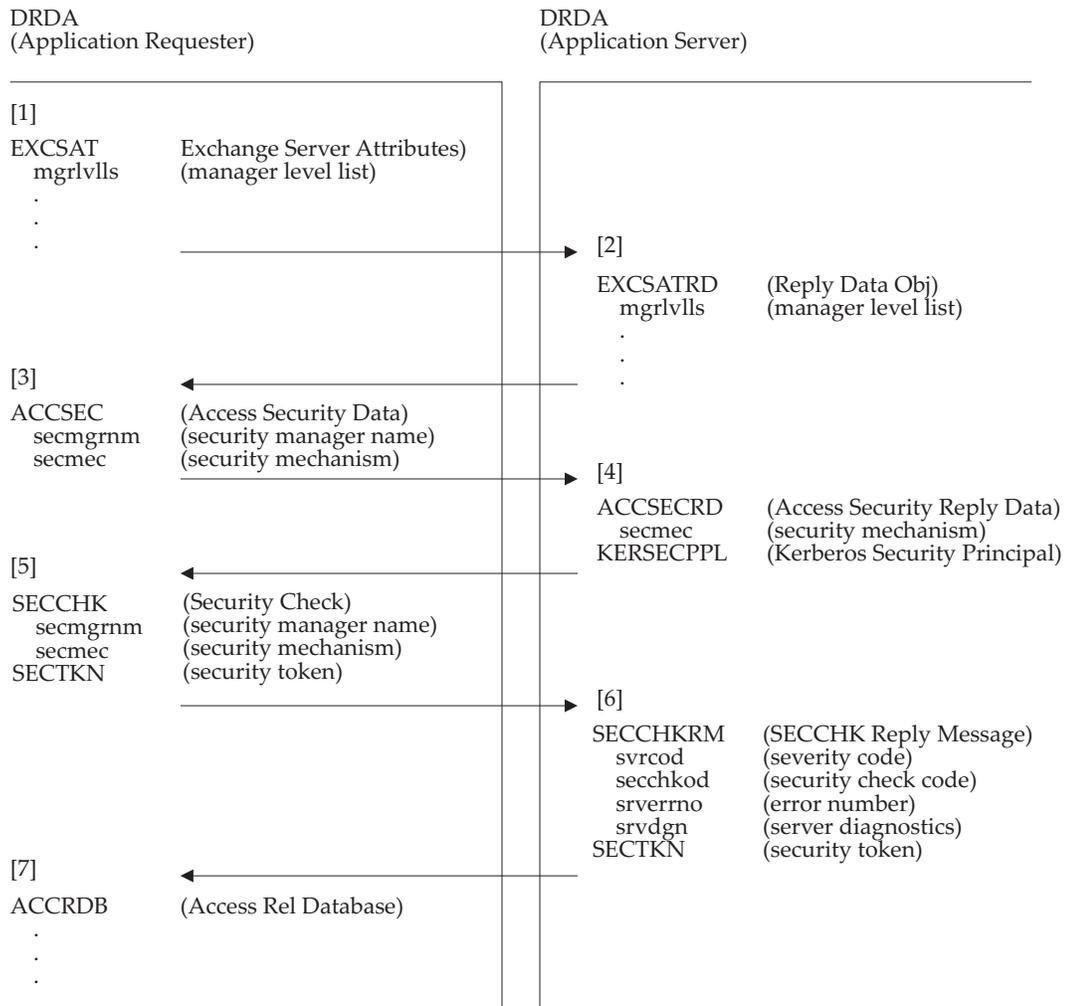


Figure 4-3 Kerberos or DCE Security Flow

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 89).

```
EXCSAT(
  mgrlvl1s(
    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(
  mgrlvl1s(
    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 97).

In this example, the application requester passes a Kerberos (*kersec*) or DCE security (*dcesec*) security mechanism 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 the ACCSEC command, the application server returns a list of security mechanism values that it does support in the *secmec* parameter on the ACCSECRD reply data object.

If Kerberos (*kersec*) is returned as a supported security mechanism inside the *secmec* within the ACCSECRD reply data object, regardless of whether *kersec* was requested on the original ACCSEC command, it is highly recommended but nonetheless not mandatory that the server also return its Kerberos principal in a KERSECPPL reply data object following the ACCSECRD. This is because the Kerberos principal for the server is required by the application requester for generating the encrypted ticket that it later needs to send to the server. If the server does not return its Kerberos principal to the application requester in reply to the ACCSEC command, the application requester may not have an alternate means of obtaining this information and may therefore be unable to make use of the Kerberos security mechanism even though it is supported by both parties.

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 user ID and password mechanism and user ID 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 42505 indicating a security verification failure.

Figure 4-4 shows the password encryption and substitution flows:

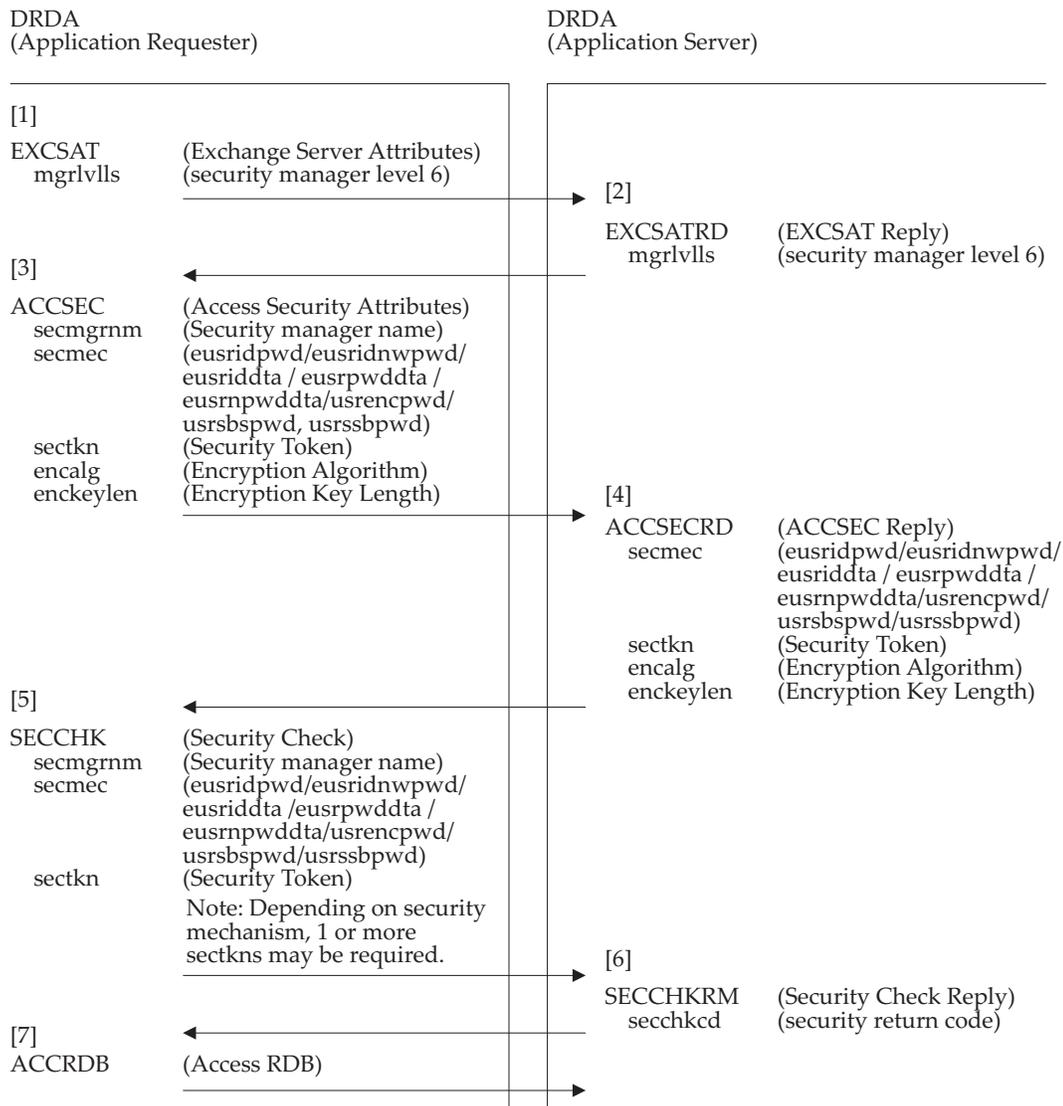


Figure 4-4 Encryption and Substitution Flow

1. The application requester specifies a SECMGR at Level 7 on the EXCSAT command and requests the use of DRDA for identification and authentication. SECMGR Level 7 is required to support the ENCALG and ENCKEYLEN instance variables on the ACCSEC command and the ACCSECRD reply.
2. The application server processes the EXCSAT command and builds a reply data object. If the ENCALG and ENCKEYLEN instance variables are supported on the ACCSEC command, SECMGR Level 7 is returned; otherwise, only security mechanisms supported by SECMGR Level 6 can be used to authenticate.
3. The application requester processes the reply. If the SECMGR is at Level 7, the application server supports the ENCALG and ENCKEYLEN instance variables. The application requester builds the ACCSEC command with the desired security mechanism, encryption

algorithm, and encryption key length.

- *eusridpwd*, *eusridnwpwd*, *usrencpwd*, *eusriddta*, *eusrpwddta*, and *eusnnpwddta*

A random large number  $x$  is generated and is used to generate the connection key  $X$  where  $X$  is equal to  $(g^x \bmod n)$ .<sup>†</sup>

- *usrbspwd* for usr ID and password substitution
- *usrsspwd* for user ID and strong password substitution

The application requester's connection key is generated. The application requester's connection key is passed in the SECTKN object.

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 appropriate reply data is generated for the requested security mechanism.

- *eusridpwd*, *eusridnwpwd*, *usrencpwd*, *eusriddta*, *eusrpwddta*, and *eusnnpwddta*

A random large number  $y$  is generated and is used to generate the connection key  $Y$  where  $Y$  is equal to  $(g^y \bmod n)$ .<sup>28</sup>

The application requester's connection key is used to generate the encryption seed  $k$  where  $k$  is equal to  $(X^Y \bmod n)$ .

- *usrbspwd* and *usrsspwd*

The application server's encryption seed is generated.

The application server's 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 appropriate SECTKN object or objects, depending on the security mechanism.

- *eusriddta*

The SECTKN is built with the encrypted user ID.

- *eusridnwpwd*

Three SECTKNs are built. The first contains the encrypted user ID, the second contains the encrypted password, and the third contains the encrypted new password.

- *eusridpwd*

Two SECTKNs are built. The first contains the encrypted user ID, the second contains the encrypted password.

- *eusnnpwddta*

Three SECTKNs are built. The first contains the encrypted user ID, the second contains the encrypted password, and the third contains the encrypted new password.

---

† 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.

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

- *eusrpwddta*

Two SECTKNs are built. The first contains the encrypted user ID, and the second contains the encrypted password.

- *usrencpwd*

One SECTKN is built containing the encrypted password.

- *usrbspwd*

One SECTKN is built containing the substitute password.

- *usrssbpwd*

One SECTKN is built containing the strong substitute password.

For encryption security mechanisms, the received application server's connection key is used to generate the encryption seed  $k$  where  $k$  is equal to  $(Y^x \text{ mod } n)$ . The token is encrypted using the encryption algorithm, the encryption seed, and the encryption token as specified by the combination of ENCALG and ENCKEYLEN parameters. For substitution security mechanisms, the substitute is generated using the application requester and application server seeds.

6. The application server processes the SECCHK command depending on the security mechanism.

- *eusriddta*

The SECTKN is decrypted to obtain the clear text user ID.

- *eusridnpwd*

Each of the three SECTKNs are decrypted to obtain the clear text user ID, password, and new password.

- *eusridpwd*

Each of the two SECTKNs are decrypted to obtain the clear text user ID and password.

- *eusrnpwddta*

Each of the three SECTKNs are decrypted to obtain the clear text user ID, password, and new password.

- *eusrpwddta*

Each of the two SECTKNs are decrypted to obtain the clear text user ID and password.

- *usrencpwd*

The SECTKN is decrypted to obtain the clear text password.

- *usrbspwd*

The SECTKN is used to generate a substitute password.

- *usrssbpwd*

The SECTKN is used to generate a strong substitute password.

For encryption security mechanisms, the token is decrypted using the encryption algorithm, the encryption seed, and the encryption token as specified by the combination of ENCALG and ENCKEYLEN parameters. For substitution security mechanisms, the

substitute is generated using the application requester and application server seeds.

The user ID, password, and potentially a new 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.

For security-sensitive data encryption security mechanisms, EUSRIDDTA, EUSRPWDDTA, and EUSRNPWDDTA, the security-sensitive objects are encrypted and decrypted using the encryption seed and the encryption token generated during the connect processing. Refer to [Section 4.4.2.2](#) for details about data encryption.

Figure 4-5 shows the plug-in security flows.

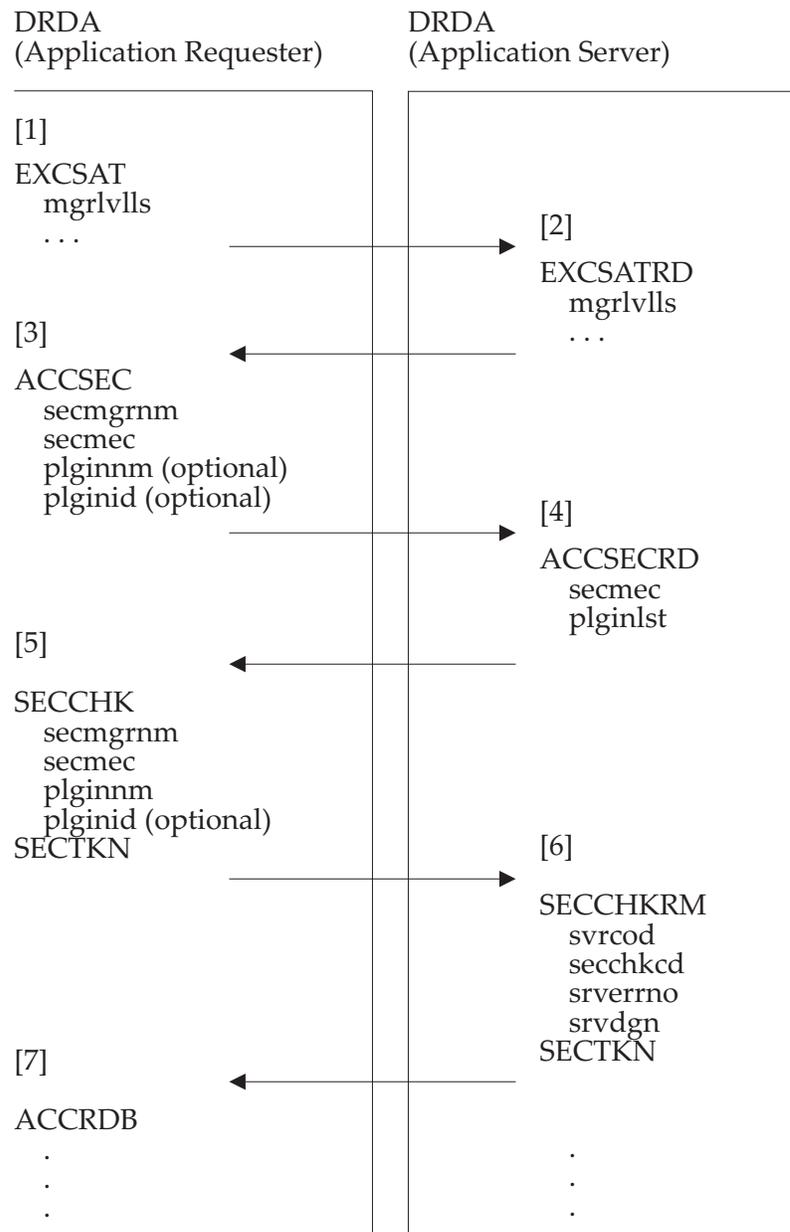


Figure 4-5 Plug-In Security Flows

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 a SECMGR at manager Level 7 on EXCSAT when requesting the use of DRDA flows for identification and authentication.

```
EXCSAT (
    MGRLVLS (
        MGRLVL ( SECMGR , 7 )
        .
        .
        . ) )
```

2. The application server receives the EXCSAT command and builds a reply data object with the SECMGR at manager Level 7 indicating it can operate at that security level. The application server sends the EXCSATRD reply data to the application server.

```
EXCSATRD (
    MGRLVLLS (
        MGRLVL ( SECMGR , 7 )
        .
        .
        . ) )
```

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

The SECMEC parameter indicates the security mechanism to use. The SECMEC values per security mechanism mapping are defined in [Table 4-3](#) (on page 97). Optionally, the PLGINNM parameter may be used to indicate a preferred plug-in module to use if the SECMEC is specified as PLGIN. The PLGINID may also optionally be used to further identify the plug-in.

4. The application server receives the ACCSEC command. It supports the security mechanism identified in the SECMEC parameter and returns the value in the ACCSECRD. If it also supports the plug-in module specified in PLGINNM, then it will return a plug-in list in PLGINLST containing a single entry whose PLGINNM is the same as in the request, and a service principal name in PLGINPPL if applicable, and optionally the plug-in version information in PLGINID. If no PLGINNM was received or the specified plug-in module is not supported, then a plug-in list containing an ordered list (from highest to lowest preference) is returned in the PLGINLST. Furthermore, the list must contain at least one entry.

If the application server does not support or accept the security mechanism specified in the SECMEC parameter on the ACCSEC command, or if the SECMEC is PLGIN and is supported by the application server but the PLGINNM is unspecified or unrecognized, the application server returns the security mechanism values that it does support in the SECMEC parameter in the ACCSECRD object. If the SECMEC includes PLGIN, then the ACCSECRD must also contain a PLGINLST listing the supported plug-in modules from highest to lowest preference and there must be at least one entry.

5. The application requester receives the ACCSECRD object and chooses the first plug-in in the PLGINLST list that it supports. The chosen plug-in generates the security token containing the security context information required for security processing using the associated principal name, PLGINPPL, if one was returned and the plug-in requires it.

The actual process to generate the security context information is not specified by DDM. The application requester may either generate the security context information, or it may call a security resource manager to generate the security context information.

The application requester passes the security context information in a SECCHK command with a SECTKN object. For information about the plug-in security context information, see

PLGINSECTKN in the DDM Reference.

6. The application server receives the SECCHK and SECTKN and uses the values to perform end-user authentication and other security checks.

The actual process to verify the security context information is not specified by DDM. The application server may either process the security context information itself or it may call a security resource manager to process the security context information.

The application server generates a SECCHKRM to return to the application requester. The SECCHKCD parameter identifies the status of the security processing. A failure to authenticate the end-user results in the SVRCOD parameter value being set to be greater than WARNING. Furthermore, if a token is generated by the application server or the security resource manager in conjunction with the failure, then it will be passed back to the application requester in a SECTKN object.

7. The application requester receives the SECCHKRM.

Assuming security processing is successful, the application requester sends a data access starting command to the application server.

SECCHKCD values indicating a failure processing the security information (for example, bad context information, expired context information) require that the security be retried or the network connection be terminated. If a token is received in the SECTKN, then it will be provided to the security resource manager to processing, regardless of the SECCHKCD value.

If security processing fails, the application requester might attempt recovery before returning to the application. For example, if the context information has expired, 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 to the application with an error indicating a security verification failure.

Figure 4-6 shows the N-flow security authentication.

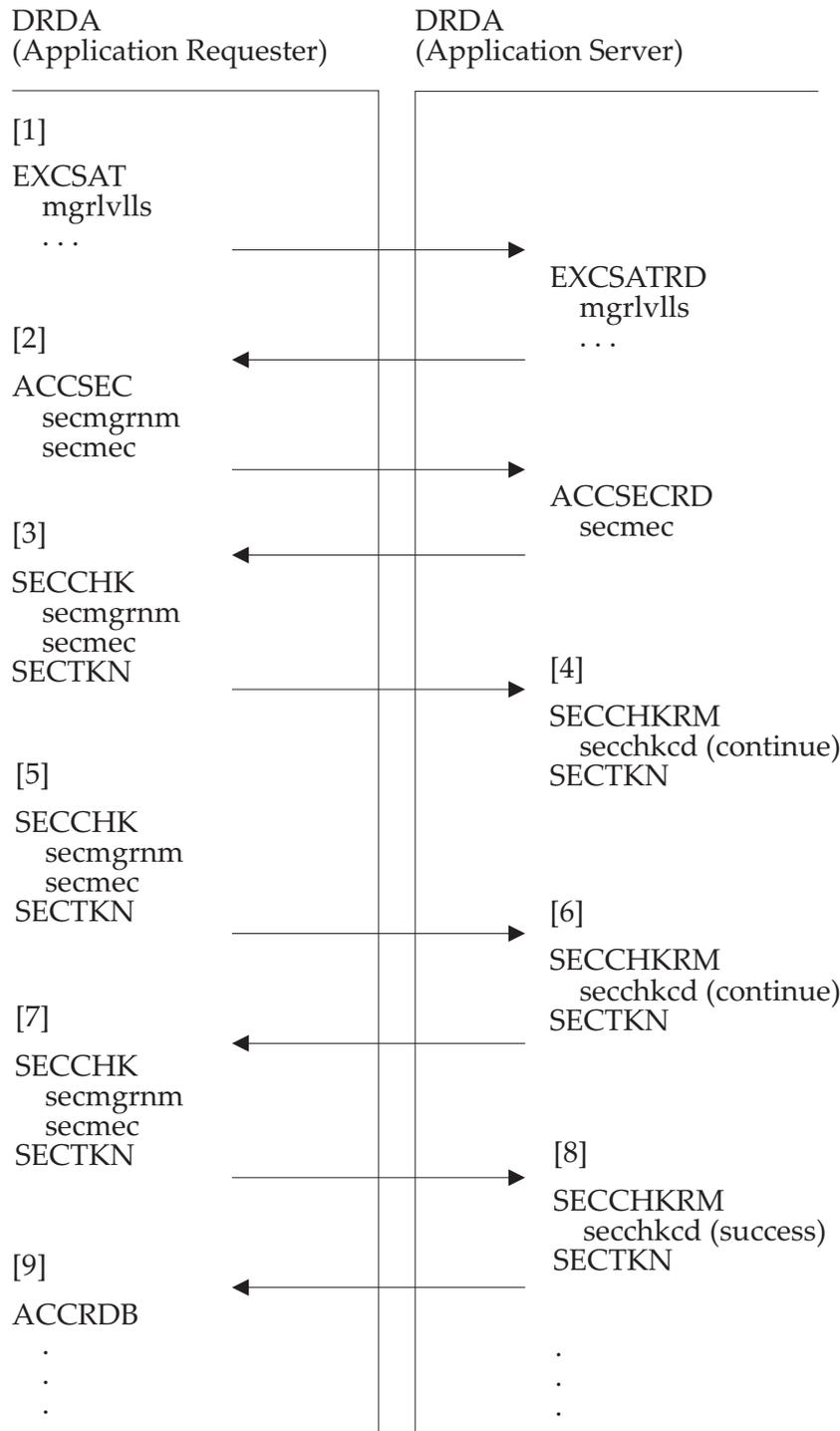


Figure 4-6 N-Flow Security Authentication

1. The application requester begins by requesting SECMGR Level 7 support in order to perform N-Flow Authentication. The SECMGR Level 7 is sent with the EXCSAT command. The application server responds with the EXCSATRD command. If N-Flow authentication is supported, the application server also returns SECMGR Level 7 within the EXCSATRD command.
2. The application requester then begins negotiation for the type of identification and authentication mechanisms through the ACCSEC command. The SECMEC parameter indicates the security mechanism to use. The SECMEC values per security mechanism mapping are defined in Table 4-3 (on page 97). The application server receives the ACCSEC command. If it supports the security mechanism identified in the SECMEC parameter, then the application server returns the value in the ACCSECRD. If the application server does not support or accept the security mechanism specified in the SECMEC parameter on the ACCSEC command, the application server returns the security mechanism values that it does support in the SECMEC parameter in the ACCSECRD object.
3. Once the security mechanism is decided upon, the application requester then begins the N-Flow negotiation of security context information. The actual process to generate the security context information is not specified by DDM. The application requester may either generate the security context information, or it may call a security resource manager to generate the security context information.

The application requester sends a SECCHK command with the security context information in a SECTKN object.

4. The application server receives the SECCHK and SECTKN and uses the values to perform end-user authentication and other security checks. The actual process to verify the security context information is not specified by DDM. The application server may either process the security context information itself or it may call a security resource manager to process the security context information.

The application server generates a SECCHKRM to return to the application requester. The SECCHKCD parameter identifies the status of the security processing. If the application server requires more security context information from the application requester, it sets up a *secchkcd* value to indicate continue. Furthermore, if a token is generated by the application server or the security resource manager, then it will be passed back to the application requester in a SECTKN object.

5. The application requester receives the SECCHKRM, and checks the SECCHKCD value to determine whether the application server requires more security context information. If so, the application requester obtains the security context information by either generating the security context information, or it may call a security resource manager to generate the security context information. The application requester sends a SECCHK command with the security context information within a SECTKN object.
6. Step 4 is repeated.
7. Step 5 is repeated.
8. The application server receives the SECCHK and SECTKN and uses the values to perform end-user authentication and other security checks. The actual process to verify the security context information is not specified by DDM. The application server may either process the security context information itself or it may call a security resource manager to process the security context information.

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. A value other than continue reflects that the server is done and

requires no more security context information. Attempts to send more security context information will result in a Protocol Violation.

Furthermore, if a token is generated by the application server or the security resource manager, then it will be passed back to the application requester in a SECTKN object.

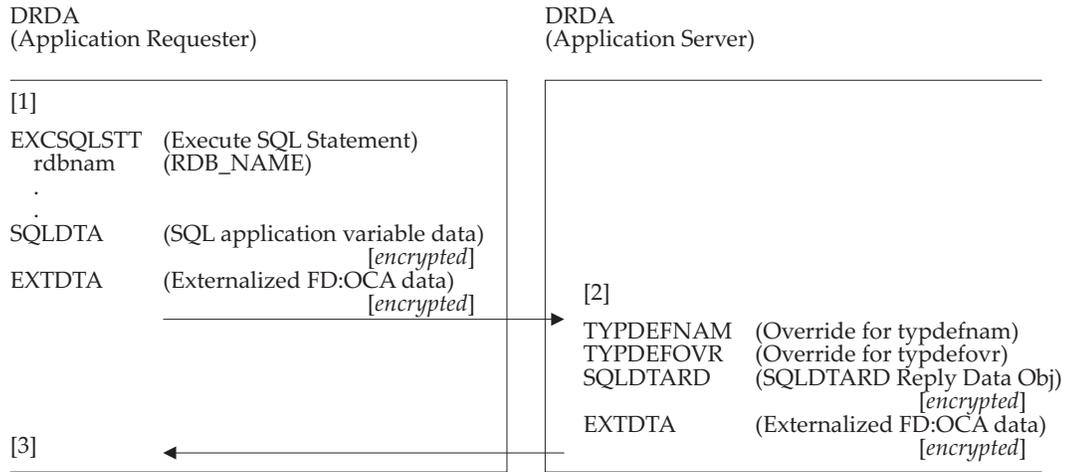
9. The application requester is now ready to access the RDB.

4.4.2.2 Security-Sensitive Data Encryption Security Flow

**Example: Application Requester and Application Server Processing**

The flows in this section show an example of the DDM commands and replies that flow during the execution of an SQL statement, when security-sensitive FD:OCA objects are encrypted.

Figure 4-7 shows an example of the security-sensitive data encryption security flows for application requester and application server processing.



**Figure 4-7** Security-Sensitive Data Encryption: Example for Requester Server Processing

The following is a discussion of the operations and functions the application requester and the application server perform for encrypting and decrypting security-sensitive data. See the EXCSQLSTT flow for a detailed description of the command flow. This section provides a brief description of some of the parameters for the DDM commands. See the DDM Reference for a detailed description of the parameters. For details of the list of security-sensitive data, see the DDM Reference, EDTASECOVR.

1. After the application requester and the application server have established the proper connection (Security Mechanism — EUSRIDDTA or EUSRPWDDTA or EUSRNPWDDTA with 56-bit DES, described in Figure 4-4 (on page 101)), a prebound SQL statement referenced in a package in the remote relational database is executed. The application requester that is acting as the agent for the application performing the execute SQL statement function creates the Execute SQL Statement command.

The application requester puts any application variable values and their descriptions, if any, in the SQLDTA command data object. For each input host variable that is a LOB data type, the application requester places an FD:OCA Generalized String header in SQLDTA and the corresponding value bytes are sent in an EXTDTA following the SQLDTA. The application requester encrypts the DSS carrier for SQLDTA and EXTDTA FD:OCA objects using the encryption seed and the encryption token generated during connect processing.

The application requester sets the *dsstype*, Encrypted OBJDSS to indicate that the object encapsulated in the DSS is encrypted.

2. The application server receives and processes the EXCSQLSTT command. The application server decrypts the encrypted security-sensitive objects using the encryption seed and the encryption token.

The application server creates SQLDTARD if any data is to be returned by the application server. For each output host variable that is a LOB data type, an FD:OCA Generalized String header is placed in the SQLDTARD and the corresponding value bytes are returned in an EXTDTA following the SQLDTARD. The application server encrypts the DSS carrier for SQLDTARD and EXTDTA objects using the encryption seed and the encryption token.

The application server sets the *dsstype*, Encrypted OBJDSS to indicate that the object is encrypted.

3. The application requester decrypts the encrypted security-sensitive objects using the encryption seed and the encryption token.

**Example 1: Intermediate Server Processing for Security-Sensitive Objects**

The flows in this section show an example of the DDM commands and replies that flow during the execution of an SQL statement, which involves intermediate server processing for encrypted security-sensitive data, using SECTKNOVR.

The flows described in this section are optional for intermediate server processing, when the intermediate server choose not to decrypt and re-encrypt the security-sensitive data by the use of the SECTKNOVR object.

Figure 4-8 shows the security-sensitive data encryption security flows for intermediate server processing using SECTKNOVR.



**Figure 4-8** Security-Sensitive Data Encryption: Intermediate Server Processing Using SECTKNOVR

The following is a discussion of the operations and functions the application server, the intermediate server, and the database server perform for security-sensitive data using SECTKNOVR. See the EXCSQLSTT flow for a detailed description of the command flow. This section provides a brief description of some of the parameters for the DDM commands. See the DDM Reference for a detailed description of the parameters. For details on the list of security-sensitive data, see the term EDTASECOVR in the DDM Reference.

1. After the application server and the intermediate server have established the proper connection (Security Mechanism — EUSRIDDTA or EUSRPWDDTA or EUSRNPWDDTA) (described in Figure 4-4 (on page 101)), a prebound SQL statement referenced in a package in the remote relational database is executed. The application server that is acting as the agent for the application performing the execute SQL statement function creates the Execute

SQL Statement command.

If the request contains SQLDTA or EXTDTA command data, then the application server encrypts the DSS carrier for SQLDTA and EXTDTA FD:OCA objects and sets the *dsstype* to Encrypted OBJDSS to indicate that the object encapsulated in the DSS is encrypted.

2. The intermediate server receives the EXCSQLSTT command and checks the *dsstype* in the DSS header to determine whether the objects are encrypted.

If the security-sensitive objects are encrypted and if the intermediate server does not want to decrypt and re-encrypt the data, then the intermediate server generates a SECTKNOVR. The intermediate server sends the encryption seed and the encryption token used for data encryption in SECTKNOVR, encryption seed SECTKN first and encryption token SECTKN second. The intermediate server encrypts the SECTKNOVR DSS using the encryption token and the encryption seed exchanged with the downstream site.

If the intermediate server receives an encrypted SECTKNOVR along with the encrypted object DSS and the downstream server connection has the same encryption requirements, then the intermediate server decrypts the SECTKNOVR using the encryption token and the encryption seed exchanged with the upstream site. The intermediate server then re-encrypts the SECTKNOVR using the encryption token and the encryption seed exchanged with the downstream site.

3. The database server receives and processes the EXCSQLSTT command. The database server first decrypts the SECTKNOVR using the encryption seed and the encryption token. The database server then decrypts the encrypted object DSS using the encryption seed in the first SECTKN of SECTKNOVR and the encryption token in the second SECTKN of SECTKNOVR.

If the reply contains SQLDTARD or EXTDTA reply data, then the database server encrypts the DSS carrier for SQLDTARD and EXTDTA FD:OCA objects. The database server sets the *dsstype*, Encrypted OBJDSS to indicate that the object encapsulated in the DSS is encrypted.

4. The intermediate server receives the reply data objects and determines whether the objects are encrypted, by checking the *dsstype* in the DSS header.

If the security-sensitive objects are encrypted, then the intermediate server generates a SECTKNOVR. The intermediate server sends the encryption seed and the encryption token used for encrypting the objects in SECTKNOVR, encryption seed SECTKN first and encryption token SECTKN second. The intermediate server encrypts the SECTKNOVR DSS using the encryption token and the encryption seed exchanged with the upstream site.

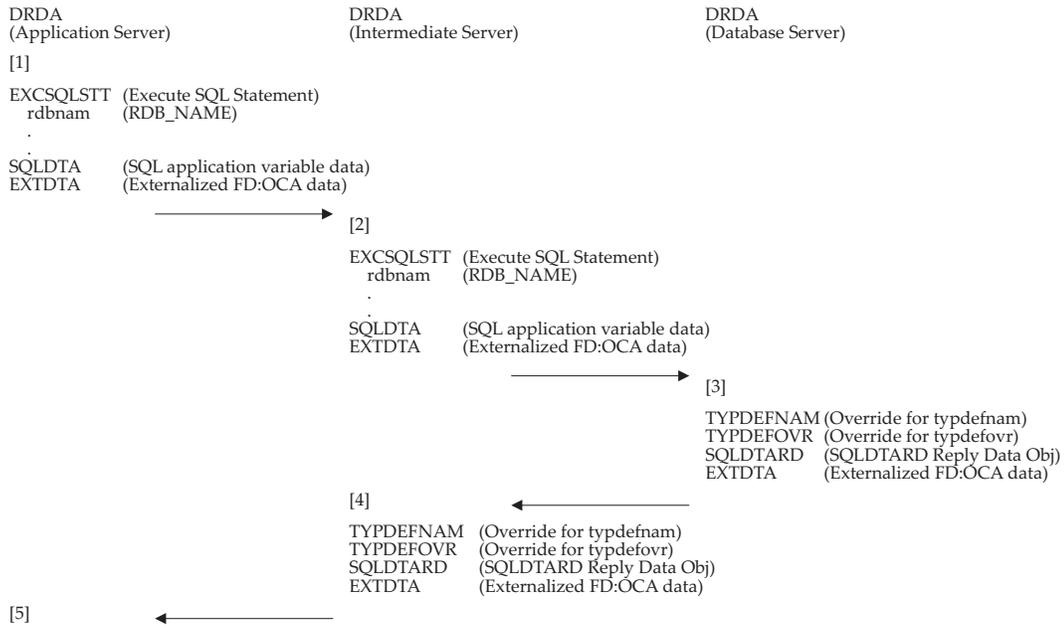
If the intermediate server receives an encrypted SECTKNOVR along with the encrypted reply objects, then the intermediate server decrypts the SECTKNOVR using the encryption token and the encryption seed exchanged with the downstream site. The intermediate server then re-encrypts the SECTKNOVR using the encryption token and the encryption seed exchanged with the upstream site.

5. The application server receives and processes the reply data object. The application server first decrypts the SECTKNOVR using the encryption seed and the encryption token. The application server then decrypts the encrypted reply object DSS using the encryption seed in the first SECTKN of SECTKNOVR and the encryption token in the second SECTKN of SECTKNOVR.

**Example 2: Intermediate Server Processing for Security-Sensitive Objects**

The flows in this section show an example of the DDM commands and replies that flow during the execution of an SQL statement, which involves intermediate server processing for encrypted security-sensitive data, where the data is decrypted and re-encrypted according to the requirements of the connection.

Figure 4-9 shows the security-sensitive data encryption security flows for an intermediate server where the intermediate server decrypts and re-encrypts the data:



**Figure 4-9** Security-Sensitive Data Encryption: Example for Intermediate Server Processing

The following is a discussion of the operations and functions the application server, the intermediate server, and the database server perform for decrypting and re-encrypting the security-sensitive data. See the EXCSQLSTT flow for a detailed description of the command flow. This section provides a brief description of some of the parameters for the DDM commands. See the DDM Reference for a detailed description of the parameters. For details on the list of security-sensitive data, see the term EDTASECOVR in the DDM Reference.

1. After the application server and the intermediate server have established the proper connection (Security Mechanism — EUSRIDDTA or EUSRPWDDTA or EUSRNPWDDTA) (described in Figure 4-4 (on page 101)), a prebound SQL statement referenced in a package in the remote relational database is executed. The application server that is acting as the agent for the application performing the execute SQL statement function creates the Execute SQL Statement command.

If the request contains SQLDTA or EXTDTA command data, then the application server encrypts the DSS carrier for SQLDTA and EXTDTA FD:OCA objects and sets the *dsstype* to Encrypted OBJDSS to indicate that the object encapsulated in the DSS is encrypted.

2. The intermediate server receives the EXCSQLSTT command and checks the *dsstype* in the DSS header to determine whether the objects in the command data are encrypted.

If the security-sensitive objects are encrypted and if the intermediate server chooses not to

use SECTKNOVR, then the intermediate server decrypts and re-encrypts the data.

If the security-sensitive objects are encrypted and if the downstream server requires a different encryption algorithm or different encryption key length, then the intermediate server decrypts and re-encrypts the objects.

If the security-sensitive objects are encrypted and if the downstream server does not require encryption, then the intermediate server decrypts the encrypted objects and sends the SQLDTA and EXTDTA in clear text.

3. The database server receives and processes the EXCSQLSTT. If security-sensitive objects are encrypted, the database server decrypts the encrypted objects. If the reply contains SQLDTARD or EXTDTA reply data, the server encrypts the objects and sets the *dsstype* in the DSS header to Encrypted OBJDSS to indicate that the object encapsulated in the DSS is encrypted.

If encryption is not required, then the database server sends the SQLDTARD and EXTDTA in clear text.

4. The intermediate server receives the reply data object. The intermediate server determines whether the objects in the reply data are encrypted, by checking the *dsstype* in the DSS header.

If the security-sensitive objects are encrypted and if the intermediate server chooses not to use SECTKNOVR, then the intermediate server decrypts and re-encrypts the data.

If the security-sensitive objects are encrypted and if the intermediate server determines that the encryption algorithm or encryption key length negotiated with the upstream server is different, then the intermediate server decrypts and re-encrypts the objects.

If the security-sensitive objects are not encrypted and if the intermediate server determines that the objects are to be encrypted before sending to the upstream server, then the intermediate server encrypts the DSS carrier for the SQLDTARD and EXTDTA objects using the encryption token and the encryption seed exchanged with the upstream site.

The *dsstype* in the DSS header is set to Encrypted OBJDSS to indicate that the object encapsulated in the DSS is encrypted.

5. The application requester decrypts the encrypted security-sensitive objects using the encryption seed and the encryption token.

4.4.2.3 Intermediate Server Processing Security Flow for Security-Sensitive Data Encryption

The flows in this section indicate the DDM commands and replies that flow during the execution of an SQL statement, which involves intermediate server processing for encrypted FD:OCA data. If encrypted FD:OCA objects are present, the first intermediate server is responsible for encrypting the encryption seed and the encryption token used to encrypt the FD:OCA objects. Every intermediate server, other than the first intermediate server, is responsible for decrypting and re-encrypting the encryption seed and the encryption token used to encrypt the FD:OCA objects.

Figure 4-10 shows the data encryption security flows for intermediate server processing.

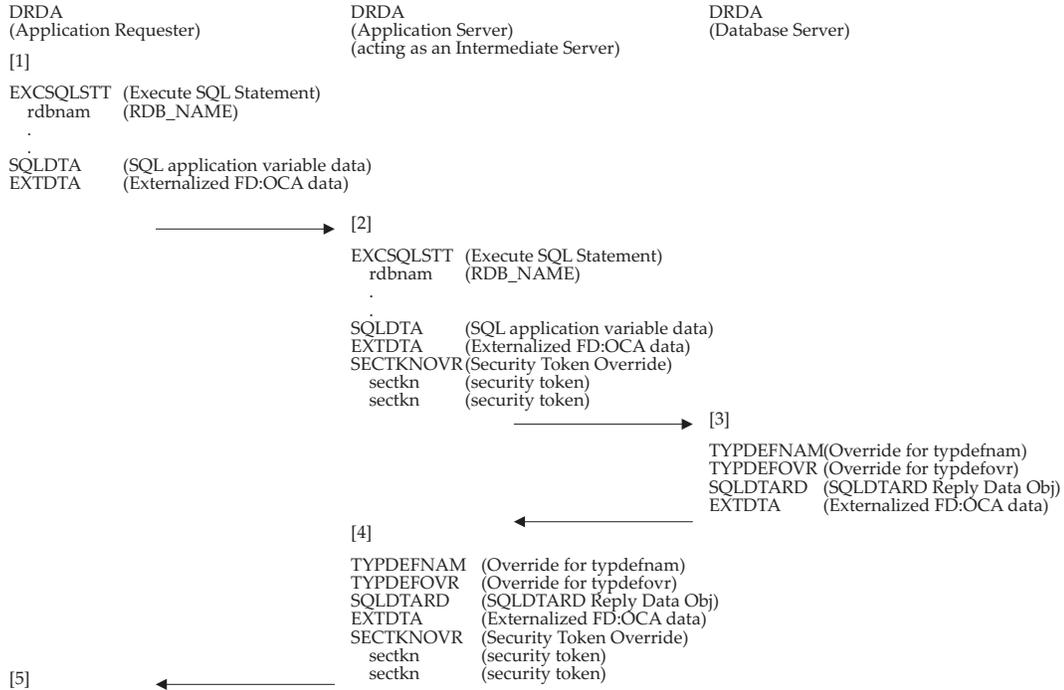


Figure 4-10 Intermediate Server Security-Sensitive Data Encryption and Decryption

1. After the application requester and the application server have established the proper connection (Security Mechanism - EUSRIDDTA or EUSRPWDDTA with 56-bit DES, described in Figure 4-4 (on page 101)), prebound SQL statements referenced in a package in a remote relational database can be executed. The application requester that is acting as the agent for the application performing the execute SQL statement function creates the Execute SQL Statement command.

If the request contains SQLDTA or EXTDTA command data, then the application requester encrypts the entire FD:OCA objects in the SQLDTA and EXTDTA carrier objects using the encryption token for the 56-bit DES encryption algorithm and the encryption seed generated from the Diffie-Hellman shared private key. The middle 8 bytes of the server’s connection key are used as the encryption token for the DES encryption algorithm with a 56-bit encryption key string value.

Similarly, if any command data flow SQLSTT, SQLDTA, or EXTDTA, then the entire FD:OCA objects in the SQLSTT, SQLDTA, and EXTDTA carrier objects are encrypted using the encryption token and the encryption seed generated from the Diffie-Hellman

shared private key.

2. The intermediate server receives the EXCSQLSTT command and determines whether the FD:OCA objects in the command data are encrypted. If the FD:OCA objects are encrypted, then the first intermediate server encrypts the encryption seed and the encryption token used for data encryption, using the encryption token and the encryption seed generated from the Diffie-Hellman private key shared with the upstream site. The security tokens are sent in SECTKNOVR, encryption seed SECTKN first and encryption token SECTKN second.

Every intermediate server, other than the first intermediate server, receives and processes the SECTKNOVR. The intermediate server decrypts the SECTKNs in SECTKNOVR using the encryption token and the encryption seed generated from the Diffie-Hellman private key shared with the downstream site. The intermediate server then re-encrypts the SECTKNs in SECTKNOVR using the encryption token and the encryption seed generated from the Diffie-Hellman private key shared with the upstream site.

3. The application server receives and processes the EXCSQLSTT command and SECTKNOVR. The application server first decrypts the SECTKNs in the SECTKNOVR using the encryption seed, and the encryption token. The application server then decrypts the encrypted FD:OCA objects using the encryption seed in the first SECTKN of SECTKNOVR, and the encryption token in the second SECTKN of SECTKNOVR.

If the reply contains SQLDTARD or EXTDTA reply data, then the application server encrypts the entire FD:OCA objects in SQLDTARD and EXTDTA carrier objects, using the encryption token and the encryption seed generated from the Diffie-Hellman shared private key. Similarly, if any reply data flows SQLDTARD, QRYDTA, EXTDTA, or SQLDARD, then the entire FD:OCA objects in the SQLDTARD, QRYDTA, EXTDTA, and SQLDARD carrier objects are encrypted using the encryption token and the encryption seed generated from the Diffie-Hellman shared private key.

4. The intermediate server receives the reply data object and determines whether the FD:OCA objects in the reply data are encrypted. If the FD:OCA objects are encrypted, then the first intermediate server encrypts the encryption seed and the encryption token used for data encryption, using the encryption token and the encryption seed generated from the Diffie-Hellman private key shared with the downstream site. The security tokens are sent in SECTKNOVR, encryption seed SECTKN first and encryption token SECTKN second.

Every intermediate server, other than the first intermediate server, receives and processes the SECTKNOVR. The intermediate server decrypts the SECTKNs in SECTKNOVR using the encryption token and the encryption seed generated from the Diffie-Hellman private key shared with the upstream site. The intermediate server then re-encrypts the SECTKNs in SECTKNOVR using the encryption token and the encryption seed generated from the Diffie-Hellman private key shared with the downstream site.

5. The application requester receives and processes the reply data object and SECTKNOVR. The application requester first decrypts the SECTKNs in the SECTKNOVR using the encryption seed and the encryption token. The application requester then decrypts the encrypted FD:OCA objects using the encryption seed in the first SECTKN of SECTKNOVR and the encryption token in the second SECTKN of SECTKNOVR.

#### 4.4.2.4 *Trusted Application Server*

The ability for an application server to access an RDB as trusted establishes a trust relationship between the application server and the database server which can then be used to provide special privileges and capabilities to an external entity such as a middleware server. When the database is accessed, a series of trust attributes are evaluated by the RDB to determine whether a trusted context is defined for the application server. The relationship between a connection and a trusted context is established when the connection is first created and that relationship remains for the life of that connection.

Refer to [Section 10.3.3](#) for more details on accessing an RDB as trusted.

#### 4.4.2.5 *Establishing a Trusted Connection*

The application server requests a trusted connection by accessing the RDB with the TRUST parameter. The RDB uses the external entity's user ID and connection trust attributes to determine whether a special relationship can be established. If a trusted context can be found that matches, a trust relationship is established. If no trust context is found or the connection does not satisfy all the trust attributes, then a warning is returned indicating that the RDB is accessed with no special capabilities and privileges. The user ID associated cannot be changed unless a complete connect flow—i.e., EXCSAT, ACCSEC, SECCHK, and ACCRDB—is processed. Refer to the following section on switching user ID associated with a trusted connection.

Once the trusted connection is established, the connection's shared private key is used for the life of the connection and is not changed when the user ID associated with the connection changes.

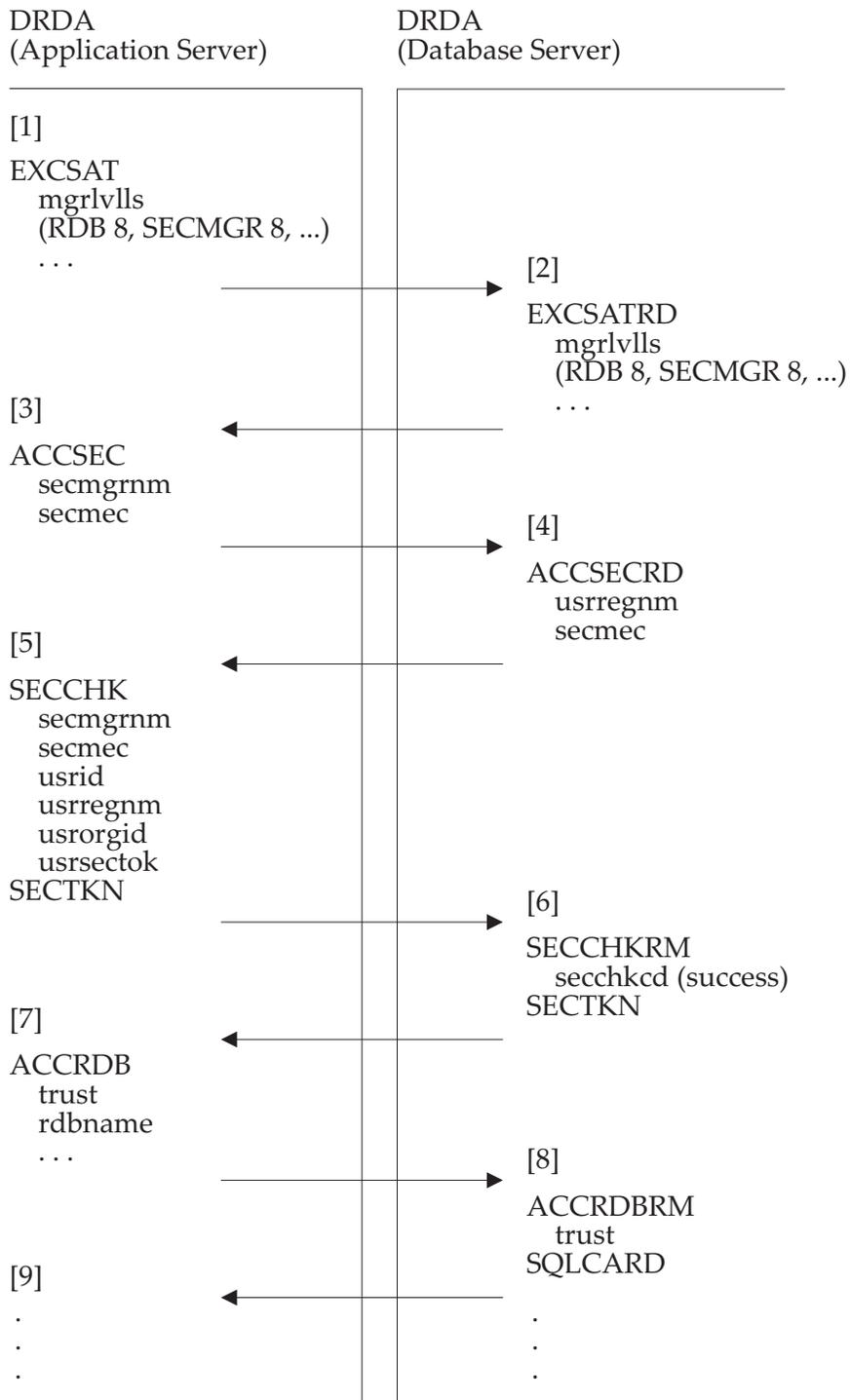


Figure 4-11 Establishing a Trusted Connection

1. The application server begins by requesting RDB Level 8 or higher to access the RDB as a trusted server. The application server can also request SECMGR Level 8 to allow identity mapping propagation between the source and target security managers. Refer to Step 3 for more details on identity mapping.

The RDB Level 8 or higher and optionally SECMGR Level 8 or higher is sent in the EXCSAT command.

2. If the RDB supports trusted connections, the database server returns RDB Level 8 in the EXCSATRD reply data. If requested and the target security manager supports identity mapping, the database server returns SECMGR Level 8 in the EXCSATRD reply data.
3. The application server begins negotiation for the type of identification and authentication mechanisms required to authenticate the system user ID of the entity. The SECCHK command with the SECMEC parameter indicates the security mechanism—and optionally the USRREGNM, USRORGID, and USRSECTOK parameters—that contains information about the USRID. A USRSECTOK contains the user's attributes such as user preferences, user groups, system privileges, or personal information. The USRREGNM contains user registry information for the USRID. The USRORGID contains the original user ID associated with the application if different than the USRID passed in the SECCHK command. The content of USRSECTOK, USRORGID, and USRREGNM is used for auditing purposes only and is not defined. The content is implementation-specific.
4. The database server receives the ACCSEC command. If it supports the security mechanism identified in the SECMEC parameter, then the database server returns the SECMEC value in the ACCSECRD. If the database server does not support or accept the security mechanism specified in the SECMEC parameter, the database server returns a list of security mechanism values that it does support in the SECMEC parameter. If the USRREGNM parameter is provided, the target security manager determines whether a relationship or an association is defined between the source user registry and the target security manager's user registry. If no mapping is defined, the database server does not return the USRREGNM indicating that no mapping exists between user registry, and all USRIDs must be defined in the target security registry.
5. Once the security mechanism and optional identity mapping are decided upon, the application server begins the authentication of the entity's system user ID of the connection. If encryption is being used on the connection, the shared private key negotiated during the establishment of the connection remains for the life of the connection. The application server sends the SECCHK command which contains the entity's USRID with its security context information in the PASSWORD parameter or in the SECTKN object.
6. The database server receives the SECCHK command and uses the values to perform authentication and other security checks. The actual process to verify the security context information is not specified. The security manager may either process the security context information itself or may call a local security resource manager to process the security context information and perform identity mapping if the USRREGNM was provided on the ACCSEC command.

The database server generates a SECCHKRM to return to the application server. The SECCHKCD parameter identifies the status of the security processing. If the database server requires more security context information from the application server, it sets up a SECCHKCD value to indicate continue. Furthermore, if a token is generated by the target security manager, then it is passed back in a SECTKN object.
7. The application server receives the SECCHKRM, and checks the SECCHKCD value to determine whether the authentication was successful. If successful, the application server sends the ACCRDB indicating that a trusted connection is requested. If more context

information is required, the authentication continues. If unsuccessful, the connection is not established.

8. The database server receives the ACCRDB command indicating a trust relationship is requested to be established between the database server and application server. The RDB searches for a unique trusted context with the required trust attributes necessary to establish the connection. The RDB algorithm used to determine whether a connection can be mapped to a trusted context is not defined by DRDA. It is up to the implementers to determine the mapping between connection attributes and trusted contexts.

If no trust context is found or the connection does not satisfy all the trust attributes, then a warning SQLCARD is returned on the ACCRDBRM with the SQLSTATE 01679 indicating that the TRUSTED CONNECTION CANNOT BE ESTABLISHED. The SQLCARD is returned with the ACCRDBRM reply and the connection is established without any extra privileges. If the entity cannot access the RDB, then the RDBATHRM is returned. If a trust context is found for the application server and the user ID, the TRUST parameter is returned with the value of TRUE; otherwise, the value of FALSE is returned.

9. The application server receives the ACCRDBRM. If successful, the connection is available for any SQL-related commands.

#### 4.4.2.6 *Switch ID Associated with a Trusted Connection*

A trusted connection provides the ability for an external entity to reuse an already established connection and switch the connection to a different user ID with a different security mechanism than was originally used to establish the connection using the entity's user ID. Depending on the security mechanism requirements of the trusted context, the user may not require any authentication credentials to flow eliminating, the need to manage end-user passwords or credentials by the application server or an external entity. The complete connection reuse flows are not required to change the user ID associated with a trusted connection. Once the new user ID is authenticated and the RDB is accessed as a trusted user, any additional privileges available to the new user under the trusted context become available. All the resources held by the previous user ID are lost. The connection state is set to the initial state as if the user ID is the first user of the connection. The previous connection state is lost. For example, any open cursor and result sets are automatically closed. Any global resources, like temp tables, are dropped. All special registers are set to the default values.

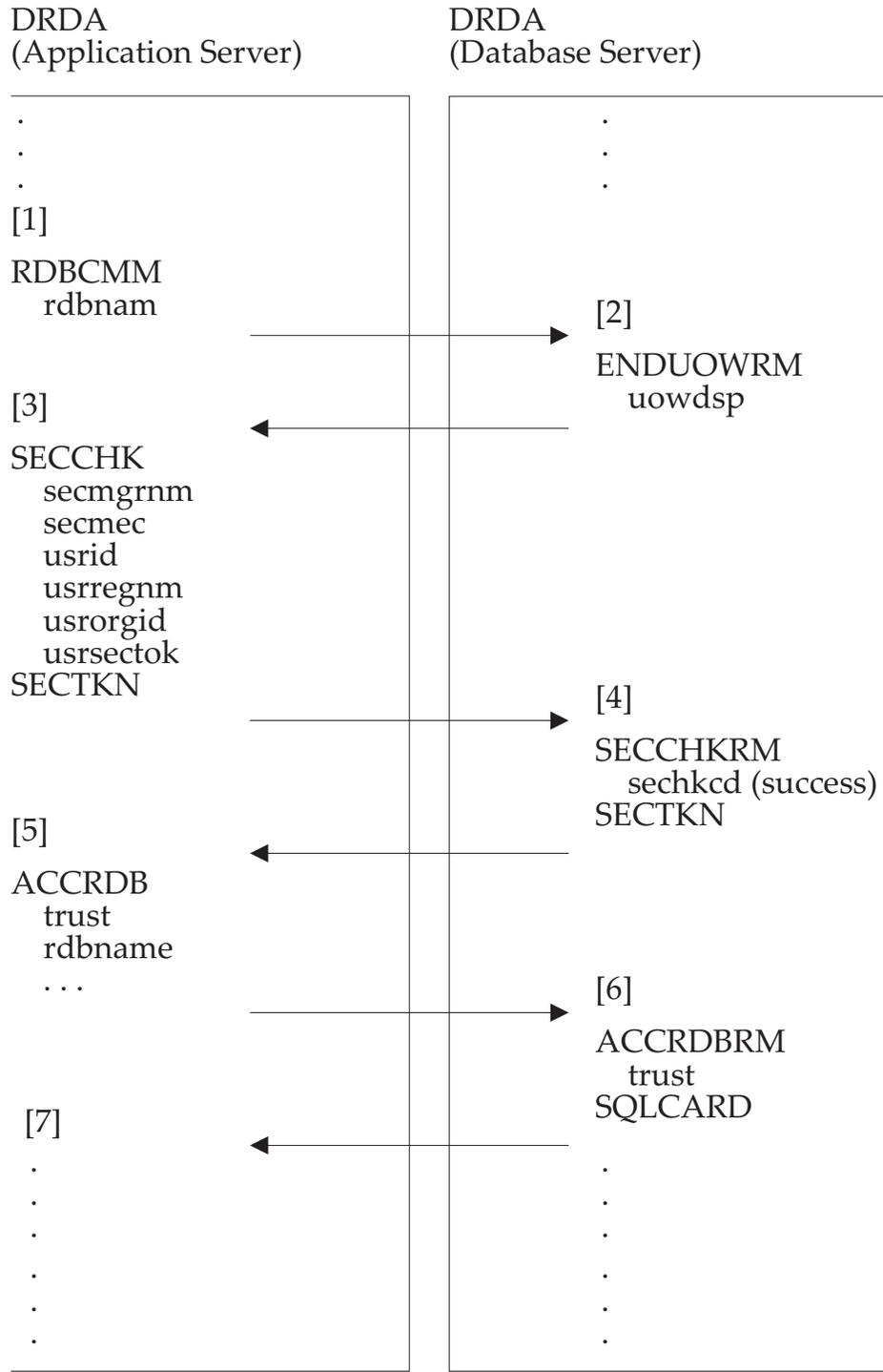


Figure 4-12 Switching ID Associated with a Trusted Connection

1. If the last command was not the ACCRDB, the application server must issue an RDBCMM command on a remote unit of work connection or a SYNCCTL command on a distributed unit of work connection to end the previous transaction.
2. The database server processes the command. The previous transaction is committed, ended, or suspended.
3. When the connection is trusted, the application server can switch the user associated with the connection by issuing a SECCHK command as long as the last command executed on the connection was a commit, rollback, or an ACCRDB command. The application server sends the SECCHK command which contains the end user's identity requesting to use the trusted connection. A USRSECTOK often contains other attributes provided by the source target security manager such as user preferences, user groups, system privileges, or personal information. The content of this field is implementation-specific. The USRREGNM contains user registry information for the USRID passed in the SECCHK command. The USRORGID contains the original user ID associated with the application if the USRID is different than that provided by the application. The content of USRSECTOK, USRORGID, and USRREGNM is used for auditing purposes only and is not defined. The content is implementation-specific.
4. The database server performs authentication depending on the security mechanism identified in the SECCHK. The actual process to verify the security context information is not specified. The security manager may either process the security context information itself or may call the local security resource manager to process the security context information and perform identity mapping which could associate a list of groups with the user ID if the USRREGNM was provided on the SECCHK command.

The database server generates a SECCHKRM to return to the application server. The SECCHKCD parameter identifies the status of the security processing. If more security context information is needed, it sets up a SECCHKCD value to indicate that authentication is continued; otherwise, it indicates whether authentication was successful or if it failed. If the connection is not trusted or the last command was not an RDBCMM, an RDBRLLBCK, a SYNCCTL, or an ACCRDB command, a PRCCNVCD protocol error is returned indicating the SECCHK command was sent in the wrong state.

5. The application server receives the SECCHKRM, and checks the SECCHKCD value to determine whether the authentication was successful. If authentication completed and was successful, the application server sends the ACCRDB indicating that the user wants to access the RDB as a trusted user. If an error occurred, the connection is not established; otherwise, authentication continues by flowing the next SECCHK command.
6. The database server receives the ACCRDB command. If the user ID is not allowed to access the RDB as a trusted user, the RDBAFLRM is returned with an SQLCARD containing the error SQLSTATE 42517, indicating the user is not authorized to use the trusted context. The connection is then put in an unconnected state. Until a valid user ID is established using the SECCHK command, all other requests are returned with an SQLCARD containing the error SQLSTATE 08003 to indicate that the command cannot be executed because the RDB is not accessed. If a trust context is found for the application server and the user ID, the TRUST parameter is returned with the value of TRUE; otherwise, the value of FALSE is returned.

If the user ID is allowed, the RDB is set to the default state. All the resources held by the previous user ID are lost. The connection state is set to the initial state. Any open cursor and result sets are automatically closed. Any global resources, like temp tables, are dropped. All special registers are set to the default values.

7. The application server receives the ACCRDBRM. If an SQLCARD is received with the error SQLSTATE 42517, it indicates that the current user ID is not allowed to use the trusted context. The connection is placed in an unconnected state. The server only accepts a SECCHK command. All other requests are returned with the SQLSTATE 08003, indicating that the command cannot be executed because the RDB is not accessed.

If successful, the ACCRDBRM is returned with the TRUST attribute set to TRUE to indicate that the application server and user ID are trusted. The RDB is ready to process SQL commands as a trusted user.

Any special privileges provided by the trusted context are now available to the user.

### 4.4.3 Performing the Bind Operation and Creating a Package

Figure 4-13 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.

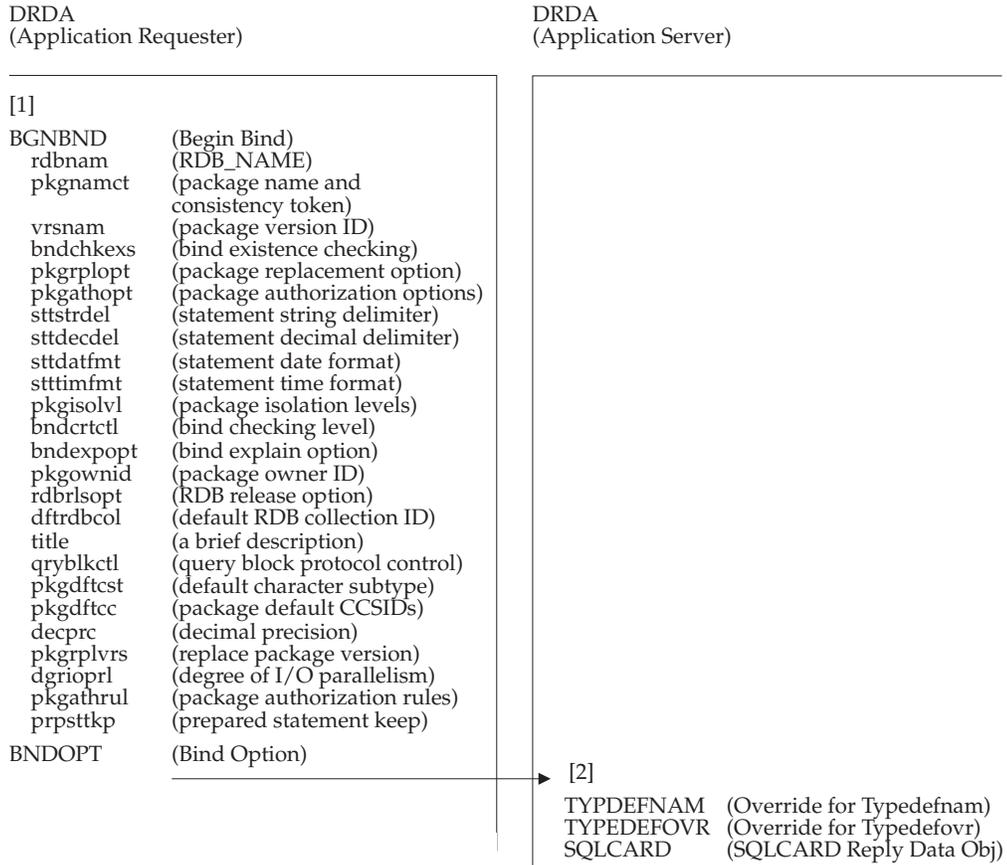
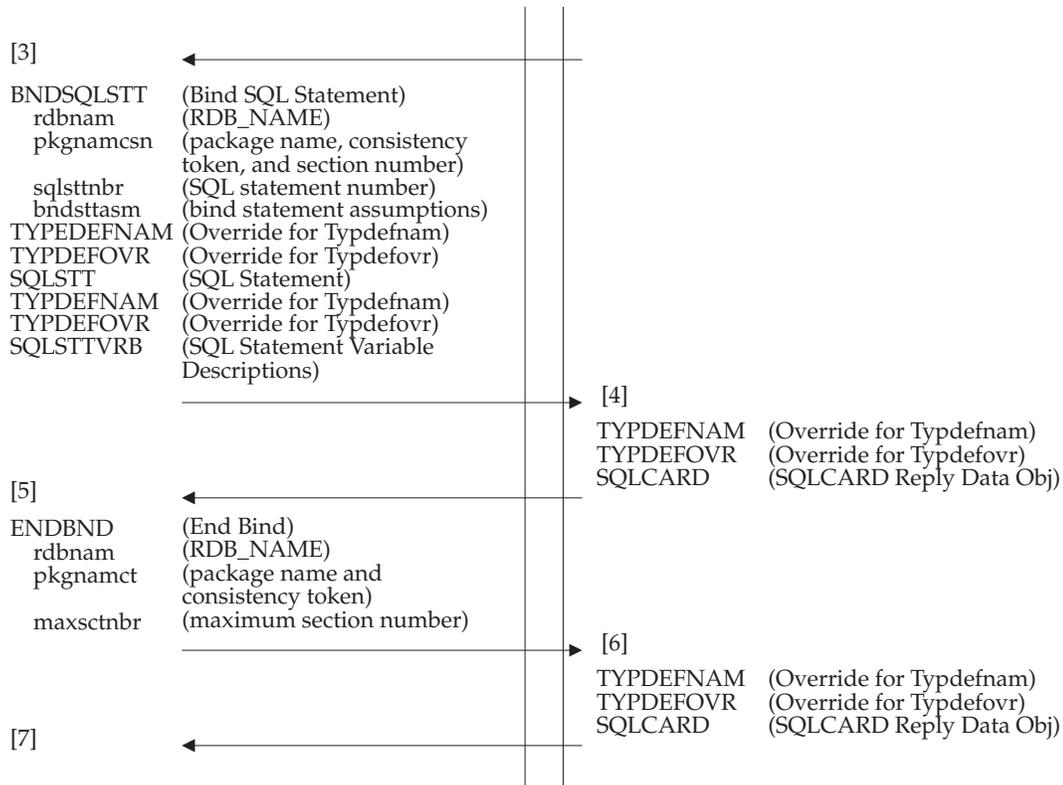


Figure 4-13 Binding and/or Package Creation (Part 1)



**Figure 4-14** 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 89)), 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 *vrnam* parameter. A null value in the *vrnam* 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<sup>29</sup> that informs the database to use I/O parallelism at the specified degree, if available.

*pkgathrul* is an optional ignorable parameter<sup>30</sup> that specifies which authorization ID should be used when executing dynamic SQL in this package.

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

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

*prpsttkp* is an optional parameter that specifies when prepared statements are released by a target RDB. The prepared statement is typically released when the work associated with it is committed or rolled back.

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 5.6.4.6](#) (on page 302).

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](#) 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 4.4.15.1](#) and [Section 4.4.15.2](#) 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 (PKGBPARAM) 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* parameter, the source application statement number in the *sqlsttnbr* parameter, whether there are statement assumptions in the *bindstasm* parameter, and a single SQL statement in the SQLSTT command data object. A detailed definition of the SQLSTT is in [Section 5.6.4.3](#) (on page 299).

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 280).

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](#) 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 7.12](#) 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 4.4.15.1](#) and [Section 4.4.15.2](#) 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.

### Overriding the Collection ID of a Package Name

A collection<sup>31</sup> can be used to group packages, and to provide package switching functionality (that is, the ability to switch between packages in different collections) to applications. The CURRENT PACKAGE PATH value, managed at the requester, is used to specify which collections should be searched, and in what order, to find a specific package. The requester modifies this value through the SET CURRENT PACKAGE PATH statement. The statement should not flow to a server during bind processing. Any new or changed value for CURRENT PACKAGE PATH is propagated to the server via the EXCSQLSET command prior to processing the next SQL request. The server will search the list to find the first collection that contains the specified package. When the CURRENT PACKAGE PATH is set, the server ignores the collection ID in the request. If the CURRENT PACKAGE PATH is not set (that is, the value is the empty string), the collection ID in the request will be used for package resolution. If the CURRENT PACKAGE PATH value is set, but the package is not part of any collection specified in CURRENT PACKAGE PATH, then the package will not be found (that is, the collection ID in the request is ignored in this case).

Prior to sending any SQL statement to a remote RDB, the requester must ensure that its CURRENT PACKAGE PATH special register value is reflected at the remote RDB. To update the value at the server, the requester uses the SQL set environment command (EXCSQLSET) specifying the new CURRENT PACKAGE PATH special register value. The server processes the EXCSQLSET command which sets the server's CURRENT PACKAGE PATH special register value. When the server processes the next SQL statement, the new CURRENT PACKAGE PATH value overrides the RDBCOLID inside any PKGNAMCSN parameter, and is used by the RDB to resolve the package's collection name. The CURRENT PACKAGE PATH value does not override the collection specified as part of the PKGNAM or PKGNAMCT parameters.

---

31. Also known in some environments as a *schema*.

#### 4.4.3.1 Perform the Bind Copy Operation to Copy an Existing Package

Figure 4-15 indicates the DDM commands and replies used to access an RDB and copy an existing package to another RDB. The result of this process is that the application requester requests the operation, but the actual copy process will occur exclusively on the application server or between the application server and another database server. If the requested copy operation is specified to occur between the application server and another database server, then a bind process takes place between the application server and the target database server who together do the syntactic and semantic checking of the SQL statements embedded in the existing package on the application server and the creation of a package on the target database server. The SQL statements and host program variables in the package which is the source of the copy are provided by the application server to create the package on another database server. An application can have multiple packages on multiple application servers and database servers.



Figure 4-15 Performing the Bind Copy Operation to Copy an Existing Package

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 requester and the server have established the proper connection (as described in [Figure 4-2](#) (on page 89)), they can perform a copy operation. The application requester and application server must support extended diagnostics to allow multiple errors to be returned with one SQLCARD. Other flows can precede or follow the copy flow and be part of the same unit of work. The requester that is acting as the agent for the application requesting the copy function creates a Copy an Existing Package (BNDCPY) command. The RDB which is to be the target of the copy operation is identified in the RDB name of the package in the *pkgnam* parameter. The source package is identified by the source collection identifier (*rdbsrcclid*), the *pkgid* in the *pkgnam* parameter, and the version name in the *vrnam* parameter. The source collection identifier, *rdbsrcclid*, like *pkgnam*, is a required parameter. A null value in the *vrnam* parameter indicates that no version ID is to be assigned to the package.

When copying a package, other bind options can be provided to override options when used in creating the source package. The BNDCPY command can also specify bind options that control various aspects of bind processing at the target server. These bind options override any bind options originally defined in the package which is the source of the copy. If the application requesting the copy wants to alter the default source for bind options that are not provided on the BNDCPY command, then an application server bind option which controls this default behavior must be provided by the application requester as a BNDOPT.

The *pkgownid* (package owner ID) and *dftrdbcol* (default collection ID) will always default to the requesting user ID if actual values are not specified in the BNDCPY command. The *pkgreplvrs* parameter can be used to specify the version name of the package to be replaced with the package at the target server.

2. The application server searches for the source package. If not found, the command is rejected with an SQLCARD. If the target server is not found, then the command is rejected with RDBNACRM with the *rdbnam* parameter containing the target server RDB name. If the source package is found, the application server and the database server establish a connection to perform a bind copy operation.

The application server that is acting as the agent for the requester performing the bind function creates a Begin Bind (BGNBND) command, 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 *vrnam* parameter. A null value in the *vrnam* parameter indicates that no version ID is to be assigned for the package. The *pkgreplvrs* parameter can be used to specify the version name of the package to be replaced with the package being bound. The BGNBND command is required to specify any bind options provided on the bind process which created the source package and any bind options provided on the BNDCPY sent by the requester.

Refer to [Section 4.4.3](#) for details on the normal DRDA bind processing scenario. The result of this process is that the application server and the database server do the syntactic and semantic checking of the SQL statements embedded in the source package and the creation of a package at the database server. The application server binds all the SQL statements and host program variables in the source package unchanged to the target package.

3. Bind errors are returned to the requester using extended diagnostics. Any non-null SQLCARDS returned from the database server that contain errors or warnings are added as conditions to the application server extended diagnostics. Diagnostics information allows

multiple errors to be returned in the SQLCA reply data. The SQLCARD returned on the ENDBND command must be the last condition appended to the extended diagnostics.

4. If the SQLCARD reply data object that is returned to the application requester indicates that the BNDCPY command was not successful, the application requester returns an exception to the application that requested the copy operation. The kind of exception that the application requester returns to the requesting application is not being architected here and is not part of DRDA. Assuming it receives a normal SQLCARD reply data object and/or after the application requester processes the SQLCARD reply data object to its satisfaction (sending any/all extended diagnostics information to the application), the application either performs other operations within the same unit of work, which can include another copy request, or it can commit or roll back the unit of work. See [Section 4.4.15.1](#) and [Section 4.4.15.2](#) for a description of commit and rollback processing in DRDA.

4.4.3.2 Perform the Bind Deploy Operation to Deploy an Existing Package

Figure 4-16 indicates the DDM commands and replies used to access an RDB and deploy an existing package to another RDB. The result of this process is that the application requester requests the operation but the actual deployment process will occur exclusively on the application server or between the application server and another database server. If the requested deployment operation is specified to occur between the application server and another database server, then a bind process takes place between the application server and the target database server who together do the syntactic and semantic checking of the SQL statements embedded in the existing package on the application server and the creation of a package on the target database server. The SQL statements and host program variables in the package which is the source of the deployment are provided by the application server to create the package on another database server.

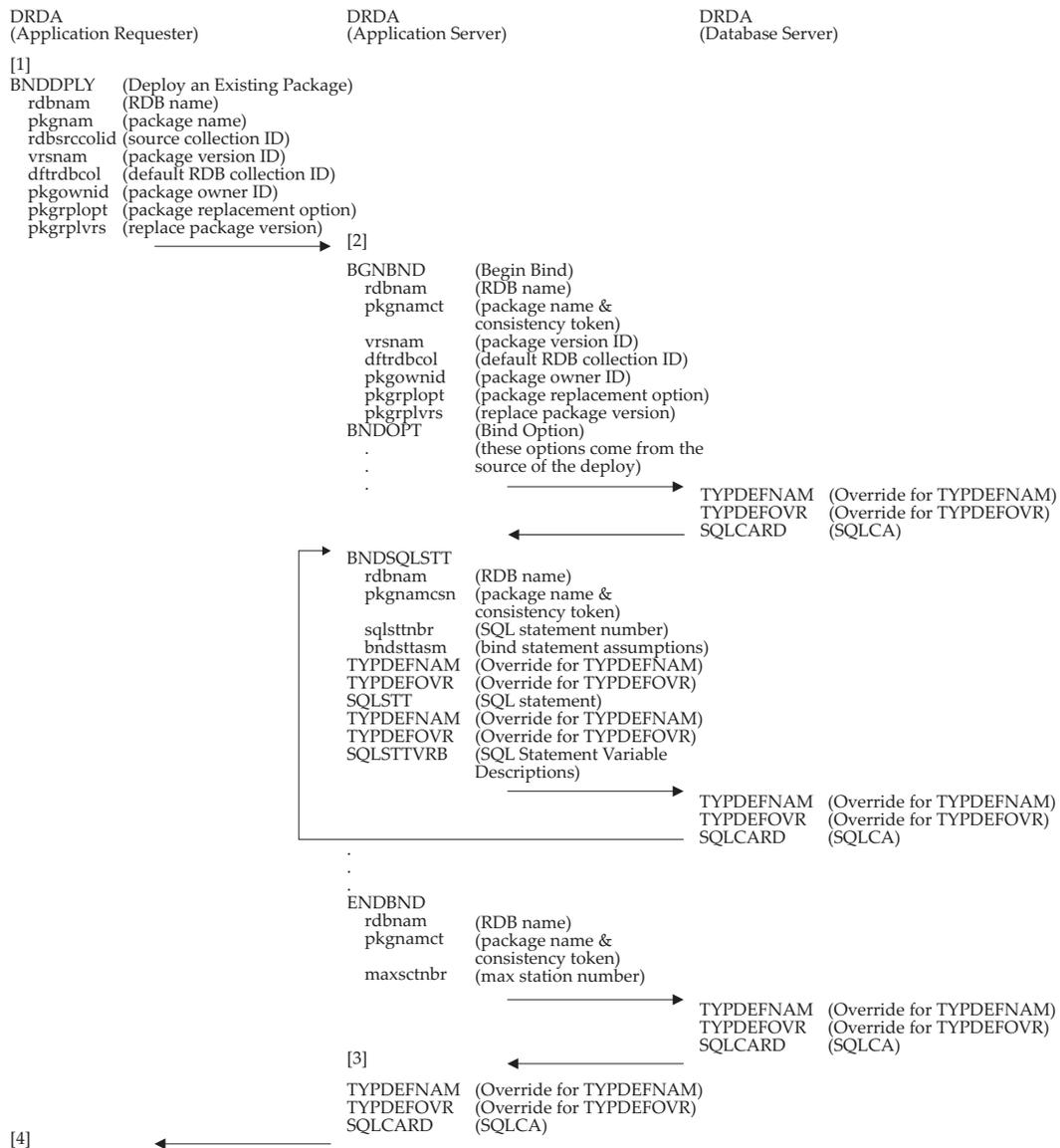


Figure 4-16 Performing the Bind Deploy Operation to Deploy an Existing Package

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 requester and the server have established the proper connection (as described in [Figure 4-2](#) (on page 89)), they can perform a deployment operation. The application requester and application server must support extended diagnostics to allow multiple errors to be returned with one SQLCARD. Other flows can precede or follow the deploy flow and be part of the same unit of work. The requester that is acting as the agent for the application requesting the deployment function creates a Deploy an Existing Package (BNDDPLY) command. The RDB which is to be the target of the deployment operation is identified in the RDB name of the package in the *pkgnam* parameter. The source package is identified by the source collection identifier (*rdbsrccolid*), the *pkgid* in the *pkgnam* parameter, and the version name in the *vrnam* parameter. The source collection identifier, *rdbsrccolid*, and version name, *vrnam*, like *pkgnam*, are required parameters. A null value in the *vrnam* parameter or the *vrnam* parameter not supplied will result in the command being rejected with a VALNSPRM.

When deploying a package, no bind options can be specified in the deployment request from the application requester other than the owner (*pkgownid*) option or the default collection ID (*dftrdbcol*). The *pkgrplors* parameter can be used to specify the version name of the package to be replaced with the package at the target server.

2. The server searches for the source package. If not found, the command is rejected with an SQLCARD. If the target server is not found, then the command is rejected with RDBNACRM with the *rdbnam* parameter containing the target server RDB name. If the source package is found, the application server and the database server establish a connection to perform a bind deploy operation.

The application server that is acting as the agent for the requester performing the deployment function creates a Begin Bind (BGNBND) command. 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 *vrnam* parameter. The *pkgrplors* parameter can be used to specify the version name of the package to be replaced with the package being bound. The BGNBND command is required to specify any bind options provided on the bind process that created the source package.

Refer to [Section 4.4.3](#) for details on the normal DRDA bind processing scenario. The result of this process is that the application server and the database server do the syntactic and semantic checking of the SQL statements embedded in the source package and the creation of a package at the database server. The application server binds all the SQL statements and host program variables in the source package unchanged to the target package.

3. Bind errors are returned to the requester using extended diagnostics. Any non-null SQLCARDS returned from the database server that contain errors or warnings are added as conditions to the application server extended diagnostics. Diagnostics information allows multiple errors to be returned in the SQLCA reply data. The SQLCARD returned on the ENDBND command must be the last condition appended to the extended diagnostics.
4. If the SQLCARD reply data object that is returned to the application requester indicates that the BNDDPLY command was not successful, the application requester returns an exception to the application that requested the deployment operation. The exception which is returned to the requesting application from the application requester is not being architected here and is not part of DRDA. Assuming it receives a normal SQLCARD reply

data object and/or after the application requester processes the SQLCARD reply data object to its satisfaction (sending any/all extended diagnostics information to the application), the application either performs other operations within the same unit of work, which can be another deployment request, or it can commit or roll back the unit of work. See [Section 4.4.15.1](#) and [Section 4.4.15.2](#) for a description of commit and rollback processing in DRDA.

#### 4.4.4 Deleting an Existing Package

Figure 4-17 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.

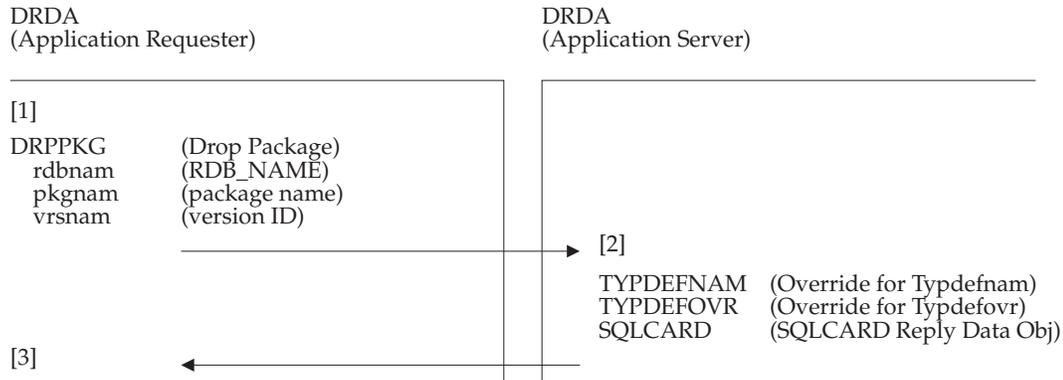


Figure 4-17 Dropping an Existing Package

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 89)). 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 4.4.15.1 and Section 4.4.15.2 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.

4.4.4.1 Delete Many Existing Packages

Figure 4-18 indicates the DDM commands and replies that flow during a process that intends to drop multiple existing packages from a relational database. The normal result of these flows is that packages which met the package qualification criteria and to which the requesting user ID is authorized will no longer exist at the remote relational database, so attempts to execute statements in those packages will now result in error conditions.



Figure 4-18 Deleting Many Existing Packages

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 requester and the server have established the proper connection (as described in [Figure 4-2](#) (on page 89)), they can perform a drop package operation which results in multiple packages at the application server being dropped. The application requester and application server must support extended diagnostics to allow multiple errors and information to be returned with one SQLCARD.

Other flows can precede or follow the deploy flow 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 criteria for which packages are desired to be dropped. If either *rdbcolidany* or *pkgidany* are provided, then *pkgnam* does not need to be provided or it will be ignored if provided. If *rdbcolidany* is provided and *pkidany* is not, then *pkgid* must be provided. If *pkgidany* is provided and *rdbcolidany* is not, then *rdbcolid* must be provided. If any version of a package is to qualify, then *orsnamany* should be provided in place of *orsnam*. 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 no packages meet the qualifications to be dropped, then the application server will return an error via the SQLCARD. For each package that met the qualification, and the requesting user had authority, an attempt will be made to remove the package from the database. If successful, the full package name will be appended as a condition to the extended diagnostics returned with the SQLCARD. If an error occurs during the dropping of any qualified package, that error will be appended as a condition to the extended diagnostics returned with the SQLCARD. As a package is successfully dropped, a savepoint will be created/replaced by the application server, and processing will continue on to the next qualified package. If an error occurs during the attempt to drop a package, the application server will roll back to the last successful savepoint, and processing will also continue on to the next qualified package. If at least one package was dropped successfully, but other errors might have occurred, then the application server will indicate that the DRPPKG command was partially successful (a warning SQLCARD will result).
3. Assuming it receives a normal SQLCARD reply data object, the application requester returns the results to the application including any extended diagnostics which will include the names of the actual package objects dropped at the application server. The application either performs other operations within the same unit of work, which can include dropping more packages, or it can commit or roll back the unit of work. See [Section 4.4.15.1](#) and [Section 4.4.15.2](#) for a description of commit and rollback processing in DRDA.

### 4.4.5 Performing a Rebind Operation

Figure 4-19 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.

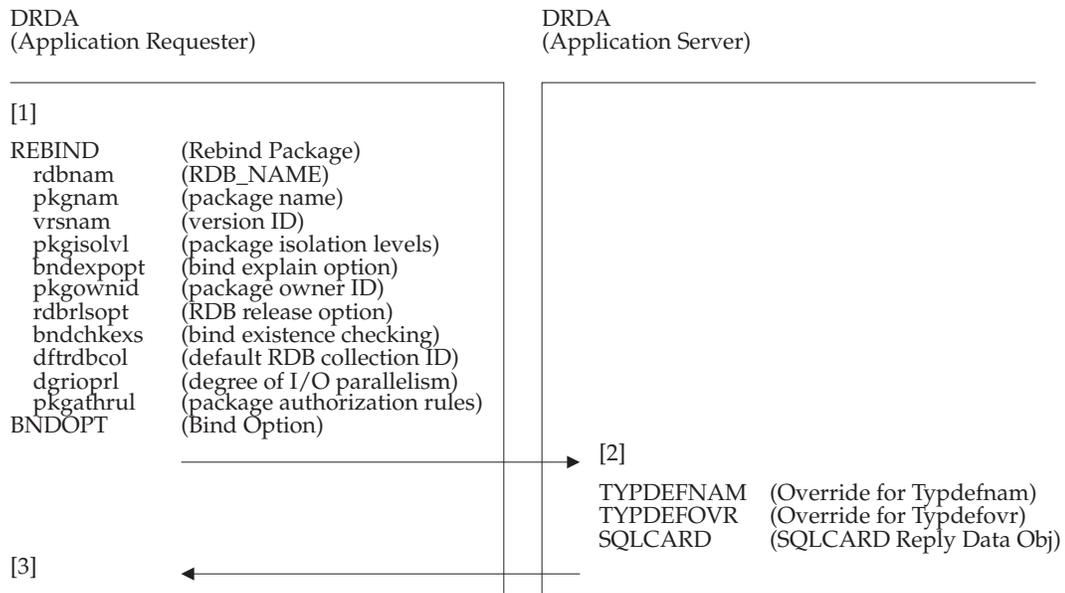


Figure 4-19 Rebinding an Existing Package

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 89)), 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.

*dgriopr1* is an optional parameter<sup>32</sup> that informs the database to use I/O parallelism at the specified degree, if available.

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

*pkgathrul* is an optional ignorable parameter<sup>33</sup> 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 5-34](#) 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 4.4.15.1](#) and [Section 4.4.15.2](#) for a description of commit and rollback processing introduced in DRDA.

---

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

#### 4.4.6 Activating and Processing Queries

Figure 4-20 and Figure 4-22 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<sup>34</sup> 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,<sup>35</sup> FIXROWPRC,<sup>36</sup> and LMTBLKPRC in the DDM Reference. Also see the rules for Section 7.22 (on page 473).

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.

The EXCSQLSET command requires package name and consistency token parameters, but no section number parameter, as it is not bound into a package. Support for the SET CURRENT PACKAGE PATH statement is contingent on support of the EXCSQLSET command, as this value is propagated from a requester to a database server (possibly through intermediate servers) using the EXCSQLSET command.

#### Requesting Describe Information

When activating a query, describe information can be requested to be returned as reply data. A light describe, a standard describe, or an extended describe may be requested. Refer to SQLDAGRP for details on the types of describe data that can be returned. The default on the open query is not to return any describe data.

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.

#### Query Instances

Consider the following scenarios involving stored procedure calls with result sets.

- **Scenario 1: Repetitive Stored Procedure Calls**

```
Application
  Call STPA          -> package DB1.TEST.STPA
    OPEN C1         -> section entry 10
  Call STPA          -> package DB1.TEST.STPA
    OPEN C1         -> section entry 10
```

---

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

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

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

- **Scenario 2: Nested Stored Procedure Calls**

```

Application
  Call STPA          -> package DB1.TEST.STPA
    OPEN C1         -> section entry 10
      Call STPA     -> package DB1.TEST.STPA
        OPEN C1    -> section entry 10
  
```

Each scenario above involves only a single application source. The *cmdsrcid* value used for each scenario serves only to identify the corresponding application source for a command. All OPEN SQL statements within each scenario map to a single query which is identified by the *cmdsrcid* as specified on the OPNQRY command.

The result of the second attempt to open cursor C1 in each example will vary. In some implementations, an error may be reported stating the cursor is already open, while in others, the cursor C1 from the previous stored procedure call may be implicitly closed to allow the subsequent open to succeed for the same cursor. Similar situations can occur with nested UDFs. Regardless, with the growing trend of applications exploiting stored procedures with result sets and UDFs, these situations can only become more common.

In order to allow a cursor to be opened again after the same cursor is already open, DRDA introduces the concept of a query instance whereby each time a cursor is opened, a unique copy or instance of the query is created. Note that, as mentioned earlier, the *cmdsrcid* is only used in identifying a query from an application source, but not a particular instance of the query. The query instance identifier is a value that gets generated by the server and returned to the application requester when the query is opened. The value of a query instance identifier is unarchitected by DRDA, the only requirement being it identifies the unique query instance. As such, an open cursor is no longer uniquely identified just by its *pkgnamcsn* and *cmdsrcid* values. Instead, the *pkgnamcsn*, the *cmdsrcid*, and the query instance identifier are required for all subsequent cursor operations (including the describe SQL statement operation and optionally the execute SQL statement for a positioned delete or update) in order to uniquely identify not only the query, but the instance thereof corresponding to the open cursor.

When a subsequent open invocation is performed on the same cursor, even though the *pkgnamcsn* and *cmdsrcid* values used have not changed compared to the previous open, the current open request will be successful, because the query instance identifier that gets generated for this particular open call will be different, thereby distinguishing this query instance from the previous one. This is the mechanism used by DRDA to support multiple open query instances for the same cursor. For more information, refer to Rules SN3, SN8, SN10, and SN11 in [Section 7.15](#) (on page 460).

In the case of an EXCSQLSTT or EXCSQLIMM command for a positioned DELETE/UPDATE SQL statement, the *cmdsrcid* for the command is used in conjunction with the cursor name as specified on the SQL statement text for uniquely identifying the query. The query instance identifier, which is optional if only a single query instance exists for the section, then uniquely identifies the instance of the query that corresponds to the referenced cursor. It should be noted that the *pkgnamcsn* specified on such an EXCSQLSTT or EXCSQLIMM command identifies the DELETE/UPDATE SQL statement as opposed to the query itself.

### Duplicate Cursors

A duplicate cursor is a cursor that gets opened even though another query instance thereof already exists as a result of a previous open on the same cursor, all within a single invocation of an application or stored procedure.

Consider the following scenario involving an application or stored procedure:

```
Application/Stored procedure
  OPEN C2 -> section entry 10
  OPEN C2 -> section entry 10
```

In this example, within a single invocation of an application or stored procedure, a second attempt is made to open cursor C2 when a query instance for this cursor already exists; that is, cursor C2 is already open. If this second open is successful, another instance of cursor C2 is created and it is considered a duplicate of the first cursor C2. Contrast this with either of the 2 scenarios in the previous section where the second instance of cursor C1 is not a duplicate cursor of the first cursor C1 since the second open takes place in a different invocation of the stored procedure.

For compatibility, the optional *dupqryok* parameter can be specified on an OPNQRY command to indicate to a server whether it should allow opening a query for a duplicate cursor. If the server is requested to open a query for a duplicate cursor but is not allowed to do so as per the setting of the *dupqryok* parameter, it must return a QRYPOPRM reply message in accordance with Rule QI2 in [Section 7.22.2](#) (on page 482).

4.4.6.1 Fixed Row Protocol

Figure 4-20 indicates the DDM commands used in the fixed row protocol query processing flows.

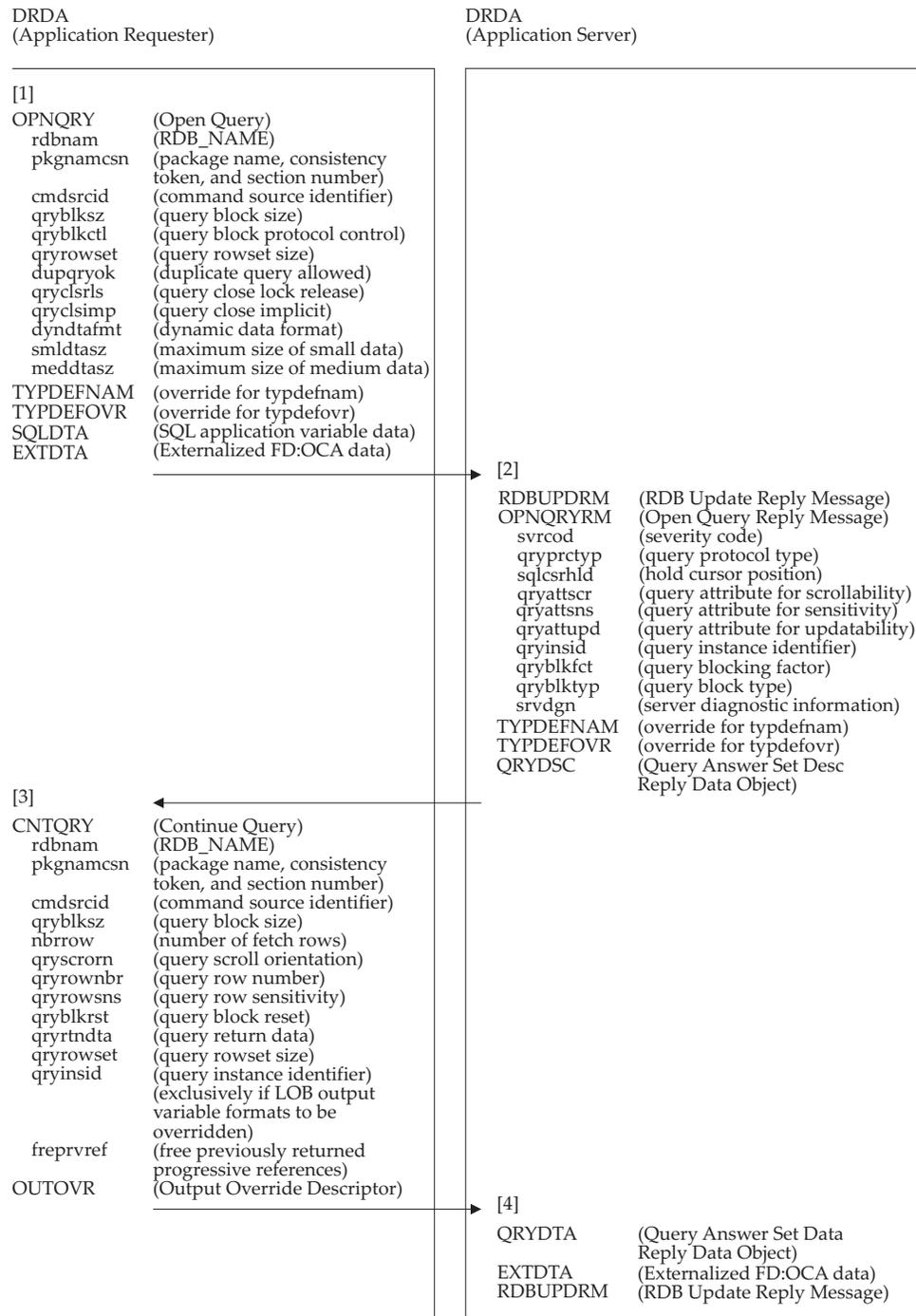


Figure 4-20 Fixed Row Protocol Query Processing (Part 1)



Figure 4-21 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.

When dealing with scrollable or rowset cursors, the explanation assumes that no *qryrowset*<sup>37</sup> parameter is specified on any OPNQRY command. For an example of how to use this parameter, refer to Section 4.4.6.2 (on page 157). For more information on scrollable cursors, refer to the Scrollable Cursor Overview description in Appendix B (on page 707).

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

37. The support for *qryrowset* on OPNQRY and EXCSQLSTT applies only to non-rowset, scrollable cursors that are not dynamic-sensitive (QRYATTNS is not equal to QRYNSSDYN). Otherwise, it is ignored. If used, it is equivalent to performing the specified number of single-row fetches across the network. Using it on these commands can provide a more efficient use of the network.

1. After the application requester and the application server establish the proper connection (described in [Figure 4-2](#) (on page 89)), an application can send 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. The application requester also indicates to the server via the optional *dupqryok* parameter whether it should allow opening a query for a duplicate cursor. The optional *cmdsrcid* parameter uniquely identifies the source of the command, which in this case is a query. The application requester 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<sup>38</sup> 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 4-13](#) (on page 127)). For the fixed row query protocol, the *qyrowset* is a parameter on the OPNQRY command that allows the requester to specify the return of a rowset with the OPNQRYRM. This parameter applies only to scrollable, non-sensitive dynamic, non-rowset cursors (QRYATTSCR is TRUE, QRYATTSNS is not QRYSNSDYN, QRYATTSET is FALSE). For these cursors, *qyrowset* specifies that a DRDA rowset is to be returned with the OPNQRYRM (see [Appendix B](#) for more information about DRDA rowsets). For all other cursors, the parameter is ignored and no data is returned with the OPNQRYRM. This example assumes that the *qyrowset* parameter is not specified. To see an example of how the *qyrowset* parameter is used, refer to [Section 4.4.6.2](#) (on page 157).

The optional *qyclsrls* parameter specifies whether read locks held by the cursor are to be released when the query is closed. The optional *qyclsimp* parameter specifies for a non-scrollable cursor whether the server should close the query implicitly when there are no more rows (SQLSTATE 02000).

The optional *dyndtafmt* parameter with the value TRUE (X'F1') requests that the application server returns FD:OCA Generalized String according to Rule DF5 (see [Section 7.8](#) for more details). The optional *smldtasz* parameter specifies the maximum size of the data in bytes to be flown in Mode X'01' of Dynamic Data Format. The optional *meddtasz* parameter specifies the maximum size of the data in bytes to be flown in Mode X'02' of Dynamic Data Format.

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 Generalized String header 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 Generalized String header 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

---

38. Also known as single row query protocols.

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 *qryblkksz* must be equal to or greater than 512 bytes and equal to or less than 10M bytes. If not, the application server returns the VALNSPRM reply message and the application server does not execute the command.

The optional *rtnsqlda* parameter controls/specifies whether describe information is returned or not; if not specified, no describe information is returned. The optional *typsqlda* parameter requests light, standard, or extended column descriptive information to be returned on the open query reply. The *rtnsqlda* parameter is not a required parameter because the QRYDSC object provides enough row-level information needed to parse the query block and return the data values to the application. If the application requires more descriptive information, the *rtnsqlda* and *typsqlda* can be specified to provide additional descriptive information such as column names, and user data type names other than the FD:OCA triplet of the base data type provided in the QRYDSC. The descriptive information is returned in the SQLDARD as reply data.

2. The application server receives and processes the OPNQRY command. The server determines that a fixed number of rows are to be returned per request because one of the following is true:
  - The application can update the rows or delete the rows through the cursor instance.
  - The cursor is scrollable and has sensitivity attributes that prevent blocking.
  - The cursor has the rowset attribute and can make multi-row fetches.

If the application server returns FD:OCA Generalized String according to Rule DF5, the *dyndtafmt* parameter is returned with the value TRUE (X'F1').

If the cursor is declared with rowset positioning, the *qryattset* parameter is returned on the OPNQRYRM. If the cursor is scrollable, the *qryattscr* parameter is returned on the OPNQRYRM.

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 which can optionally be preceded by an RDBUPDRM reply message as per the Update Control (UP Rules) in [Section 7.19](#) (on page 467). The *qryinsid* parameter on the OPNQRYRM reply message uniquely identifies the instance of the query. If the server chooses to impose a blocking factor on the query which limits the number of rows that can be blocked at a time without influence from the application requester on the OPNQRY command, the *qryblkfct* parameter will contain the blocking factor. 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 the QRYDSC reply data object that follows either an OPNQRYRM or the 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](#) gives a detailed definition of the QRYDSC.

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](#) for detail on QRYDSC.
- 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 be the first or only DSS in the reply chain. For those reply messages that require an SQLCARD, the SQLCARD reply data object, indicating the condition, follows the reply message in the reply chain that is sent.
- In response to an OPNQRY command for a query that is currently suspended (previously opened and has not been terminated), if the application server is unable to generate a unique *qryinsid* for this query instance, it returns a QRYPOPRM reply message as the first or only object in the reply chain.

If the *qyrowset* parameter applies to the cursor (in particular, is not a rowset cursor) and is specified on the OPNQRY command, the server prepares to return a DRDA rowset of the specified size to the requester. Otherwise, no data rows are returned with the OPNQRY command. Refer to [Appendix B](#) for details regarding when the *qyrowset* parameter applies. This example assumes that the *qyrowset* parameter is not specified on the OPNQRY, so no data rows are returned at this time. To see an example of how the *qyrowset* parameter on the OPNQRY and EXCSQLSTT is used, refer to [Section 4.4.6.2](#) (on page 157).

Depending on the value of the *rtnsqlda* and *typsqlda* parameters on the open query, an SQLDARD data can be generated to provide descriptive information related to the low-level row data described in the QRYDSC and returned in the QRYDTA. The level of describe information returned is identified by the value of the *typsqlda* parameter.

3. The application requester places any input variables from the application in the SQLDTA command data object. Any input variable requiring its data to be flown as externalized data generates an FD:OCA Generalized String header for the data in the SQLDTA and creates an EXTDTA object to contain the externalized data. For each of the input variables that are LOB SQL data types, the following occurs. 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.<sup>39</sup> The OUTOVR object will be rejected by the application server with an OBJNSPRM if OUTOVRROPT is set to OUTOVRNON on the OPNQRY command. 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 the one determined by the application server using Data Format (DF Rules), 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 Data Format (DF Rules); see [Section 7.8](#) for more details.

**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 (CLSQR) 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 CLSQR command successfully, it terminates the query and sends the application requester an SQLCARD reply data object indicating that it has closed the query. All the progressive references that originate from the query are also freed.

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 optional *cmdsrcid* parameter uniquely identifies the source of the command, which in this case is a query. The mandatory *qryinsid* parameter is then used to uniquely identify the instance of the query in use. The application requester reflects the application requested scrolling options and multi-row fetch operation in the *qryscrovrn*, *qryrownbr*, *qryrowsns*, *qryblkcrst*, *qryrtndta*, and *qryrowset* parameters. If the *qryattset* parameter identifies a rowset cursor, the *qryrowset* must be specified on the CNTQRY. When processing a rowset cursor, flexible blocking is used. See Block Formats (BF Rules) in [Section 7.22.1.1](#) for detail on flexible blocking.

The application requester can supply a different value in the *qryblksz* parameter.

The optional *freproref* parameter specifies whether all the previously returned progressive references in a complete row should be freed upon completion of this command.

4. The application server receives the CNTQRY command and the OUTOVR, if present. 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 externalized data in the row.

For multi-row fetch, the requester must provide a statement-level SQLCA to the application. For rowset cursors, the server returns the statement-level SQLCA for the rowset for each QRYDTA that contains the rows in the rowset. The row-level SQLCAs in the QRYDTA are set to null because the fetch for a rowset cursor is an atomic operation and

---

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

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

only one SQLCA is returned to the application. For non-rowset cursors, only row-level SQLCAs are returned in the QRYDTA.

If any FD:OCA Generalized String columns are in the row being retrieved, data will be returned as follows. All LOB locators are returned in the QRYDTA object. Each FD:OCA Generalized String column is returned according to the Data Format (DF Rules); see [Section 7.8](#) for more details. If more than one EXTDTA is returned, they are returned in the order that their corresponding FD:OCA Generalized String headers occur in the QRYDTA object. This reply chain can optionally be followed by an RDBUPDRM reply message as per the Update Control (UP Rules) in [Section 7.19](#) (on page 467).

See [Section 5.5.3.1](#) for a detailed definition of QRYDTA.

If the server sends any QRYDTA objects, then each QRYDTA is either an exact query block or a flexible query block, according to the server's requirements or preferences. The rules for these two Block Formats are given in Block Formats (BF Rules) in [Section 7.22.1.1](#) (on page 474). If the server is using exact blocking rules to generate the QRYDTA(a) to contain the query data, then:

- If a single row of the answer set data cannot be contained in a single query block of the exact size specified in the *qryblksz* parameter, 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 and is returned as a query block shorter than the specified *qryblksz* parameter (a short block).
- A row that is larger than one query block flows in multiple query blocks, each of which is a QRYDTA DSS object exactly conforming to the *qryblksz* parameter, except (potentially) the last one. The requester must pull the reply data objects back together into a single QRYDTA data object.

If the server is using flexible blocking rules to generate the QRYDTA(s) to contain the query data, then:

- If a single row of the answer set data or the answer set of a multi-row fetch (an SQL rowset) cannot be contained in a query block of the initial size specified in the *qryblksz* parameter, then the query block is expanded to the size required to contain the entire row or SQL rowset. If the query block is not full, it is truncated at the end of the data and is returned as a query block shorter than the specified *qryblksz* parameter (a short block).

In response to a CNTQRY command, if the application server is not going to send the QRYDTA reply data objects, then a DDM error reply message must be the first object in the reply chain. For those reply messages that require an SQLCARD, the SQLCARD reply data object, indicating the condition, follows the reply message in the reply chain that is sent.

The response to the previous OPNQRY command defined the data that flows to the application for this command. Unless these formats are overridden by an OUTOVR object, this is the format of the data in the query blocks.

5. The application requester receives the QRYDTA reply data object and maps the row data to the application's host variables. If there is externalized data, the application requester obtains the column value bytes from the associated EXTDTA objects that follow the QRYDTA. When performing a multi-row fetch operation for a rowset cursor, flexible blocking is used. See Block Formats (BF Rules) in [Section 7.22.1.1](#) for detail on flexible blocking.

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 optional *cmdsrid*

parameter uniquely identifies the source of the command, which in this case is a query. The mandatory *qryinsid* parameter is then used to uniquely identify the instance of the query in use. The requester reflects the application requested rowset and scrolling options in the *qryscorn*, *qryrownbr*, *qryrowsns*, *qryblkrst*, *qryrtdta*, and *qryrowset* parameters. The requester can supply a different value in the *qryblksz* parameter. It then sends a 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.

If the cursor is not closed implicitly by the server, the application requester can close the cursor by sending a CLSQRY command to the server. The optional *qryclsrls* parameter specifies whether read locks held by the cursor are to be freed. Its setting will override any previous setting that may have been specified earlier on the OPNQRY command.

- n* For non-scrolling cursors, or queries, 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 may choose to generate an ENDQRYRM reply message and send 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. The server may also choose not to close a non-scrollable cursor implicitly depending on whether it is a query with the HOLD option, and also on the value of the *qryclsimp* parameter that has previously been specified on the OPNQRY command. The application server does not close the cursor implicitly, even though the end of the query data is reached, if one of the following conditions is true:

1. There are any previously returned progressive references that belong to a complete row and are not freed, and FREPRVREF=TRUE is not specified with the CNTQRY command.
2. There are any previously returned progressive references that belong to an incomplete row.
3. There are any new progressive references to be returned upon completion of the CNTQRY command.

For cursors that scroll, a CNTQRY that runs out of rows in the answer set does not result in an ENDQRYRM and closed cursor; that is, the server does not close the query implicitly. 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, regardless of the value of the *qryclsimp* parameter on the OPNQRY command. 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 multi-row fetch, the good rows are returned with the ENDQRYRM and SQLCARD with the error indication.
- 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* 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 FD:OCA Generalized String Data in Answer Set)

The limited block protocol applies only to non-rowset cursors.

Figure 4-22 indicates the DDM commands used by the limited block query processing flows when there are no FD:OCA Generalized String outputs. Refer to Section 4.4.6.3 if there are FD:OCA Generalized String columns in the answer set.

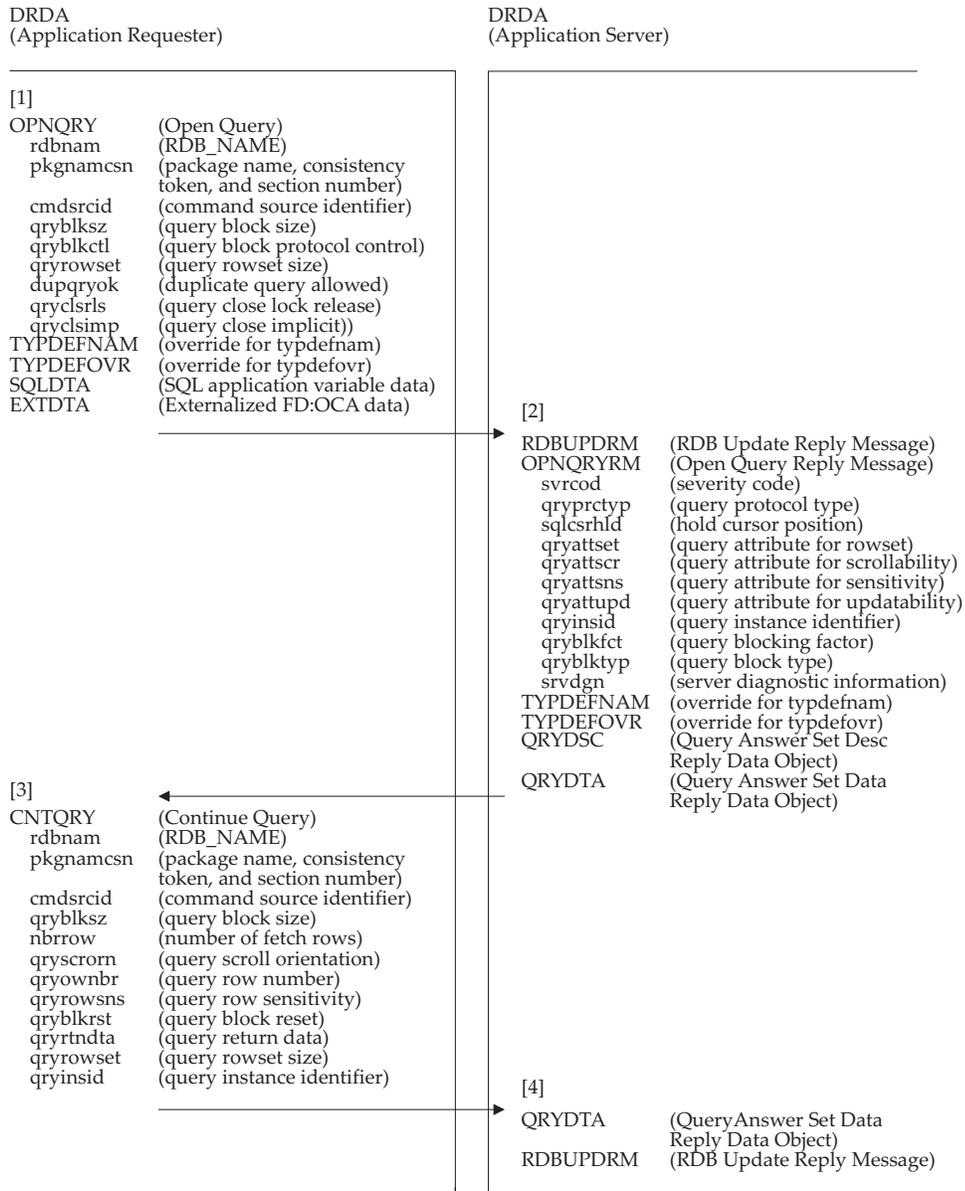
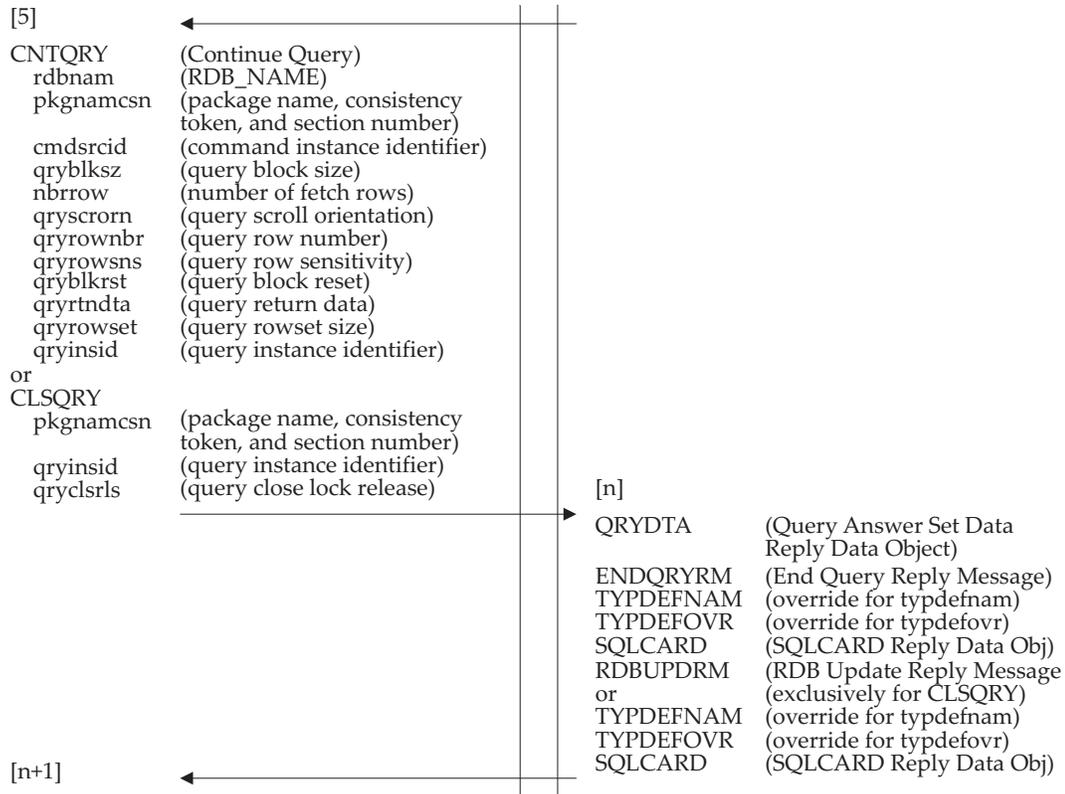


Figure 4-22 Limited Block Protocol Query Processing (No FD:OCA Generalized String Data) (Part 1)



**Figure 4-23** Limited Block Protocol Query Processing (No FD:OCA Generalized String Data) (Part 2)

The following is a discussion of the operations and functions the application requester and the application server perform.<sup>40</sup> 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 89)), 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

40. When dealing with a scrollable, non-rowset cursor, this example assumes that a *qryrowset* parameter is specified on every CNTQRY command but may or may not be specified on the OPNQRY command. Since the application requester is using the *qryrowset* parameter, there may be a difference between the cursor position known to the application and the cursor position at the target server. The application requester has two general methods for handling this difference. The first method requires the application requester to force the two cursor position values to be equivalent. The second method requires the application requester to map between the application's and the target server's cursor position. In this example, the application requester uses the second method. These and other topics concerning scrollable cursors are discussed in detail in the Scrollable Cursor Overview description in [Appendix B](#) (on page 707). A *qryrowset* parameter may also be specified for non-scrollable, non-rowset limited block protocol cursors, but since the cursor is only forward-moving these considerations do not apply.

(see *qryblkctl* in [Figure 4-13](#) (on page 127)). Its absence here allows limited block processing. The *qryrowset* parameter applies to both non-scrollable and scrollable cursors and specifies whether the indicated number of single-row fetches are to be attempted in order to return a rowset with the OPNQRYRM. This example assumes that a *qryrowset* value is not sent on the OPNQRY command. 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 FD:OCA Generalized String data types are handled as in [Section 4.4.6.1](#) (on page 148).

The application requester also indicates to the server via the optional *dupqryok* parameter whether it should allow opening a query for a duplicate cursor. The optional *cmdsrcid* parameter uniquely identifies the source of the command, which in this case is a query.

The optional *qryclsrls* parameter specifies whether read locks held by the cursor are to be released when the query is closed. The optional *qryclsimp* parameter specifies for a non-scrollable cursor whether the server should close the query implicitly when there are no more rows (SQLSTATE 02000).

2. The application server receives and processes the OPNQRY command. It then determines that it will use limited block protocols, for example, because the cursor is a non-rowset cursor, 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, is used in determining 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. The *qryinsid* parameter on the OPNQRYRM reply message uniquely identifies the instance of the query. If the server chooses to impose a blocking factor on the query which limits the number of rows that can be blocked at a time without influence from the application requester on the OPNQRY command, the *qryblkfct* parameter will contain the blocking factor. The OPNQRYRM can optionally be preceded by an RDBUPDRM reply message as per the Update Control (UP Rules) in [Section 7.19](#) (on page 467).

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

The application server then generates an FD:OCA data descriptor that describes an SQLCA to be returned for each row and the default format for each column in the row. This description is placed in the QRYDSC reply data object that in turn is placed in the reply chain after the OPNQRYRM reply message or warning SQLCARD.

The application server can also create a QRYDTA reply data object and place it in the reply chain after the QRYDSC reply data object. Whether the server returns query data after the QRYDSC depends in part on the requester's specification (for example, the *qryrowset* parameter), the type of data returned (FD:OCA Generalized String or not), and the server's preference.

If the server sends any QRYDTA objects, then each QRYDTA is either an exact query block or flexible query block, according to the server's requirements or preferences. The rules for these two Block Formats (BF) are given in [Section 7.22.1.1](#) (on page 474).

If the server is using exact blocking rules to generate the QRYDTA(s) containing the query data, then:

- If a single row of the answer set data cannot be contained in a single query block of the exact size specified in the *qryblksz* parameter, then it will span two or more query blocks. If the last block is not full, it can be truncated at the end of the data and

returned as a query block shorter than the specified *qryblksz* parameter (a short block).

- A row that is larger than one query block flows in multiple query blocks, each of which is a QRYDTA DSS object exactly conforming to the *qryblksz* parameter, except (potentially) the last one.
- If the server will return more than one row, then each subsequent row will fill the remaining unused space in the query block containing the end of the previous row, and the remainder of the row data will be placed in one or more additional query blocks, so that each query block returned is of the exact query block size, except (possibly) the last one.
- If a row or rows spans multiple query blocks, the requester must pull the reply data objects back together into a single QRYDTA data object.

If the server is using flexible blocking rules to generate the QRYDTA(s) to contain the query data, then:

- If a single row of the answer set data cannot be contained in a query block of the initial size specified in the *qryblksz* parameter, then the query block is expanded to the size required to contain the entire row. If the query block is not full, it can be truncated at the end of the data and returned as a query block shorter than the specified *qryblksz* parameter (a short block).
- If the server will return more than one row, then each subsequent row will fill the remaining space in the query block containing the end of the previous row. If the remaining space in the query block cannot contain the entire row, then the query block is expanded to the size required to contain the entire row.
- If the query block has been expanded, then no more rows can be added to that query block. If the server can return one or more additional rows, then it creates a additional query block of the initial size and adds rows to it as for the first query block.

If the server sends any query data, the number of rows returned depends on the *qryblksz* parameter, the actual size of each returned row, the *qyrowset* parameter (if any), and the *maxblkext* parameter (if any). If the cursor is non-scrollable and no *qyrowset* parameter is specified, then the number of rows returned depends only on the *qryblksz* and the *maxblkext* value. If the cursor is non-scrollable and a *qyrowset* parameter is specified, the server attempts to return a DRDA rowset of the size indicated. If the cursor is scrollable, the application server attempts to return either a DRDA rowset of the size indicated in the *qyrowset* parameter or an implicit rowset (according to Query Data Transfer Protocol rule QP4) when no *qyrowset* parameter was sent.

If query data is to be returned, and the data is not returned in a DRDA rowset, then at least one whole row is returned. Moreover, any other rows that can be fitted into the last query block for that row may also be returned. If extra query blocks can be returned, then additional rows can be added to extra query blocks up to the limit allowed by the *maxblkext* parameter.

If query data is to be returned, and the data is returned in a DRDA rowset, the DRDA rowset consists of the first row in the result table followed by the next rows in sequence up to the number of rows specified explicitly or implicitly by the *qyrowset* parameter. The DRDA rowset is complete when all the requested rows are returned, a FETCH request for a row results in a negative SQLCODE, or a FETCH request results in an SQLSTATE of 02000. If the application server cannot return all the requested rows because it has used up all the query blocks it is allowed to, according to the *maxblkext* parameter, then the DRDA rowset is incomplete. The application server returns one or more QRYDTAs with the rows it has fetched, and expects the CNTQRY request from the application request either to complete

the DRDA rowset (or, optionally) to reset it.

If the application server can place all of the data rows in the answer set in the last query block, then the query block is complete. The application server may chain an ENDQRYRM and associated SQLCARD to the reply chain, as indicated in Step n. If this is done, the application server terminates the query as described in Step n.

The application server then sends the reply chain to the application requester.

3. The application requester receives the OPNQRYM 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 (CLSQR) command with the correct package name, consistency token, and section number in the *pkgnamcsn* parameter, along with the optional *cmdsrcid* parameter to uniquely identify the query, and the mandatory *qryinsid* parameter to uniquely identify the instance of the query, and optionally the *qryclsrls* parameter to the application server.

If the application server processes the CLSQR 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.

When the application requests a row from the application requester, the application requester attempts to retrieve the desired row from the received query blocks (QRYDTAs), if any.

If the cursor is non-scrollable, the desired row is the next row in the received query blocks. If the cursor is scrollable, the row position may need to be evaluated against the scrolling parameters on the FETCH request to determine whether the next row satisfies the scroll requirements of the FETCH request. If not, the desired row is obtained by calculated position number within the received query blocks.

If the desired row is in the received query blocks, the row data is returned to the application. The application requester maps the row data it received in the QRYDTA reply data object(s) to the application's host variables. The data in the QRYDTA is described either by the QRYDSC reply data object returned at the beginning of query processing or by the QRYDSC as modified by the application requester's OUTOVR specification.

As the application requests each additional row from the application requester, the application requester repeats this process of retrieving the desired row from the received QRYDTA objects and returning the received data to the application.

If no query blocks were returned with the OPNQRY command, or if the desired row is not in the received QRYDTA objects, or if all of the rows in the QRYDTA objects have been received, then the application requester creates a CNTQRY command with the same value for *pkgnamcsn* as the corresponding OPNQRY command supplied, along with the optional *cmdsrcid* parameter to uniquely identify the source of the query. The mandatory *qryinsid* parameter then uniquely identifies the instance of the query. The *qryblksz* parameter can contain a different value from the OPNQRY (or previous CNTQRY) and will result in a new size for the query block(s) the application server will return. For non-scrollable cursors and for scrollable cursors that are not being accessed in a scrollable manner by the application requester, there are no further considerations. For cursors for which a *qryrowset* value was specified, the application requester must consider whether the data was part of a DRDA

rowset and whether that DRDA was complete or incomplete. If the DRDA rowset received was incomplete, the application requester must either send a request to complete or reset the pending DRDA rowset according to [Appendix B](#) (on page 707). To complete the DRDA rowset, the application requester sends a CNTQRY request by specifying a *qryrowset* value equal to the remaining number of rows in the original DRDA rowset. To reset the DRDA rowset, the application requester specifies a *qryblkrst* value of TRUE.

4. When the application server receives the CNTQRY command, it places query data in one or more query blocks according to whether it sent data with the previous command (OPNQRY or CNTQRY). If the application server did not return any query data with the previous command, then it populates one or more QRYDTA objects as described in Step 2. If the application did return any query data with the previous command, then it must take the previously returned data into account when returning additional data.
  - If exact blocking is in effect for the cursor and there is a partial row from the previous command, the application server places the remainder of the partial row in the first query block(s) before adding additional rows. Additional rows are returned according to the description in Step 2.
  - If an incomplete DRDA rowset is pending from the previous command, the application server first validates that the CNTQRY command either completes the DRDA rowset or resets it. If the DRDA rowset is to be completed, the application server only returns as many rows as are needed to complete the DRDA rowset, up to the limits set by the *maxblkext* value. It is possible that the DRDA rowset may remain pending after the CNTQRY command has executed. If the DRDA rowset is to be reset, any partial row that occurs (when exact blocking is in effect) is discarded and any pending extra query blocks are discarded before a new DRDA rowset is started, using navigational and sensitivity controls specified in the CNTQRY command. See [Appendix B](#) for more details.

If the application server can place all of the data rows in the answer set in the last query block, then the query block is complete. The application server may chain an ENDQRYRM and associated SQLCARD to the reply chain, as indicated in Step n. If this is done, the application server terminates the query as described in Step n.

The reply chain can optionally be followed by an RDBUPDRM reply message as per the Update Control (UP Rules) in [Section 7.19](#) (on page 467).

The application server then sends the reply chain to the application requester.

5. When the application requester receives the QRYDTA reply data object(s), it retrieves row data from the received QRYDTA object(s) to return to the application. If exact blocking is in effect for the cursor and the row data spanned the last query block from the previous command and the query block(s) just received, then the application requester must first pull the separate pieces together to form the first row. Otherwise, the first row is obtained from the first row contained in the QRYDTA(s) returned with this command. The first row is returned to the application.

The row data for the first row is returned to the application as described in Step 3.

As the application requests each additional row from the application requester, the application requester repeats the process of retrieving the desired row from the received QRYDTA objects and returning the received data to the application.

When the desired row cannot be obtained from the received query blocks, the application requester creates a CNTQRY command as described in Step 3.

Steps 4 and 5 are repeated as long as the application can request rows from the cursor or

until the application does not request any more rows. In the case of non-scrollable cursors, the application cannot request additional rows after it has fetched the last row in the answer set.

- n* When the application server receives the CNTQRY command, it places query data in one or more query blocks as described in Step 4.

If the application server can place all of the data rows in the answer set in the last query block, then the query block is complete. If permitted by the *qryclsimp* parameter specified on the OPNQRY command, by the properties of the query itself (for example, the query is not WITH HOLD and not scrollable), and by the requirements of Chaining Rule CH5, the application server may be able to close the query if this occurs.

If it is permitted to close the query or is required to do so by the *qryclsimp* parameter, the application server places an ENDQRYRM reply message and the associated SQLCARD reply data object in the reply chain after the last query block.

If the application server adds the ENDQRYRM reply message and the SQLCARD reply data object to the reply chain, then the query is terminated. The application server closes the query and will no longer accept CNTQRY commands for this cursor until an OPNQRY is again processed for this cursor.

Otherwise, the application server does not send the ENDQRYRM and its associated SQLCARD to the application requester with this reply chain. They will be sent as the only responses to the next CNTQRY command. The application server may not close the query until a CLSQRY command is sent from the application requester, or until the transaction is rolled back.

The application server then sends the reply chain to the application requester.

- n+1* When the application requester receives the reply object(s), it retrieves row data from the received QRYDTA object(s) to return to the application as described in Step 5.

When the application requester receives the ENDQRYRM reply message, it knows that the application server has processed the last row of answer set data or a terminating error has occurred. It knows that the application server has closed the query or that the query has been terminated and is in a not-opened state, 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 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 COMMIT, through any method, causes the termination of a query, except for queries with the HOLD option in the DECLARE CURSOR section.

#### 4.4.6.3 Limited Block Protocol (FD:OCA Generalized String Data in Answer Set)

The limited block protocol applies only to non-rowset cursors.

The *rtnextdta* parameter determines the major processing flows when there are FD:OCA Generalized String columns in the answer set. The presence of a *qryrowset* parameter in this case requires that the *rtnextdta* value must be *rtnextall*. If Dynamic Data Format is enabled with `OUTOVROPT=OUTOVRNON`, then the *rtnextdta* value must be *rtnextall* as well.

Figure 4-24 and Figure 4-25 indicate the complete limited block processing flows when the *rtnextdta* value is *rtnextall*.

Figure 4-24 and Figure 4-25 also indicate partial limited block processing flows when the *rtnextdta* value is *rtnextrow*. In addition, refer to Figure 4-26 for additional commands that apply in this case.

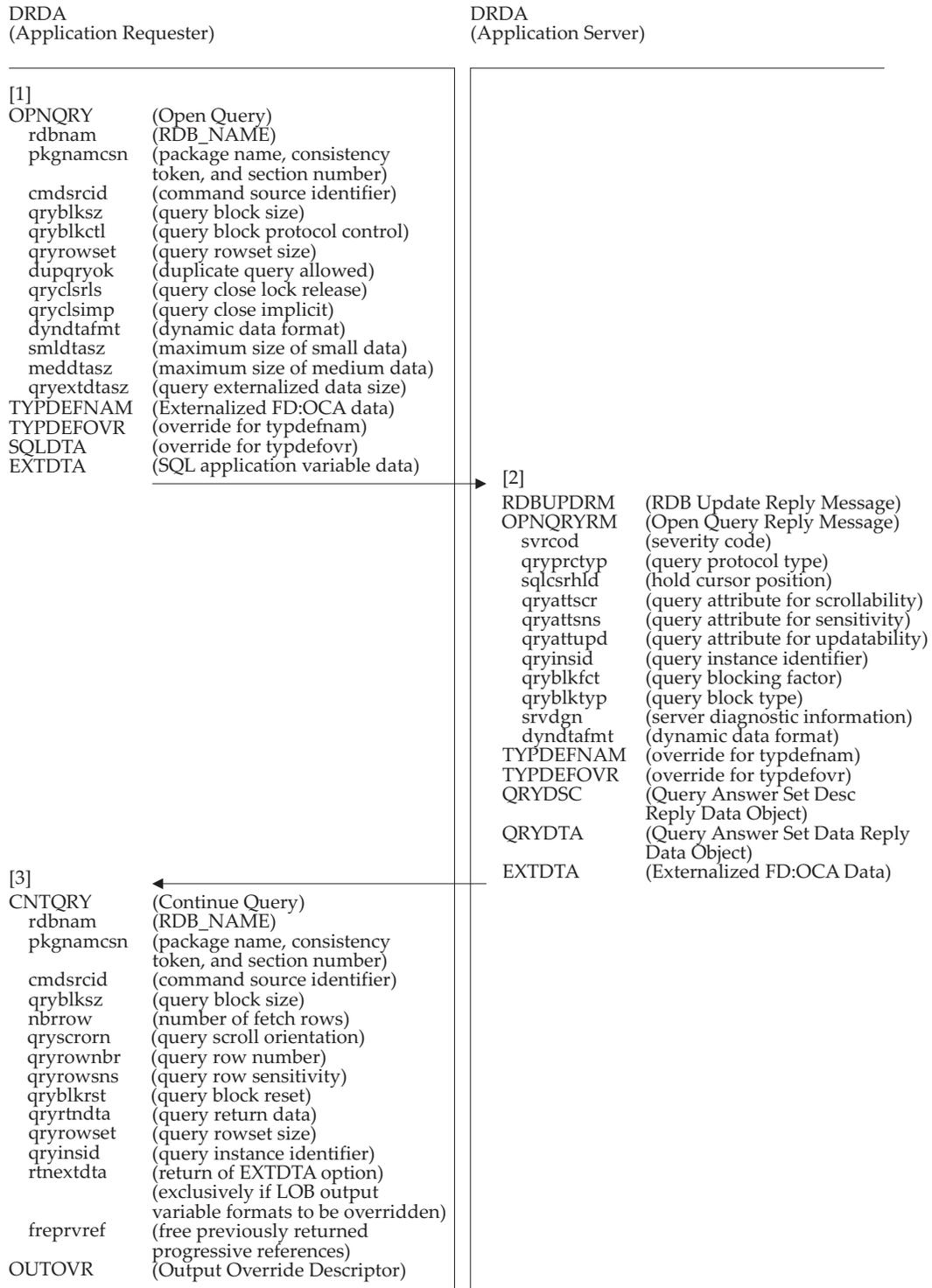


Figure 4-24 Limited Block Protocol Query Processing (FD:OCA Generalized String Data, rtnextall) (Part 1)

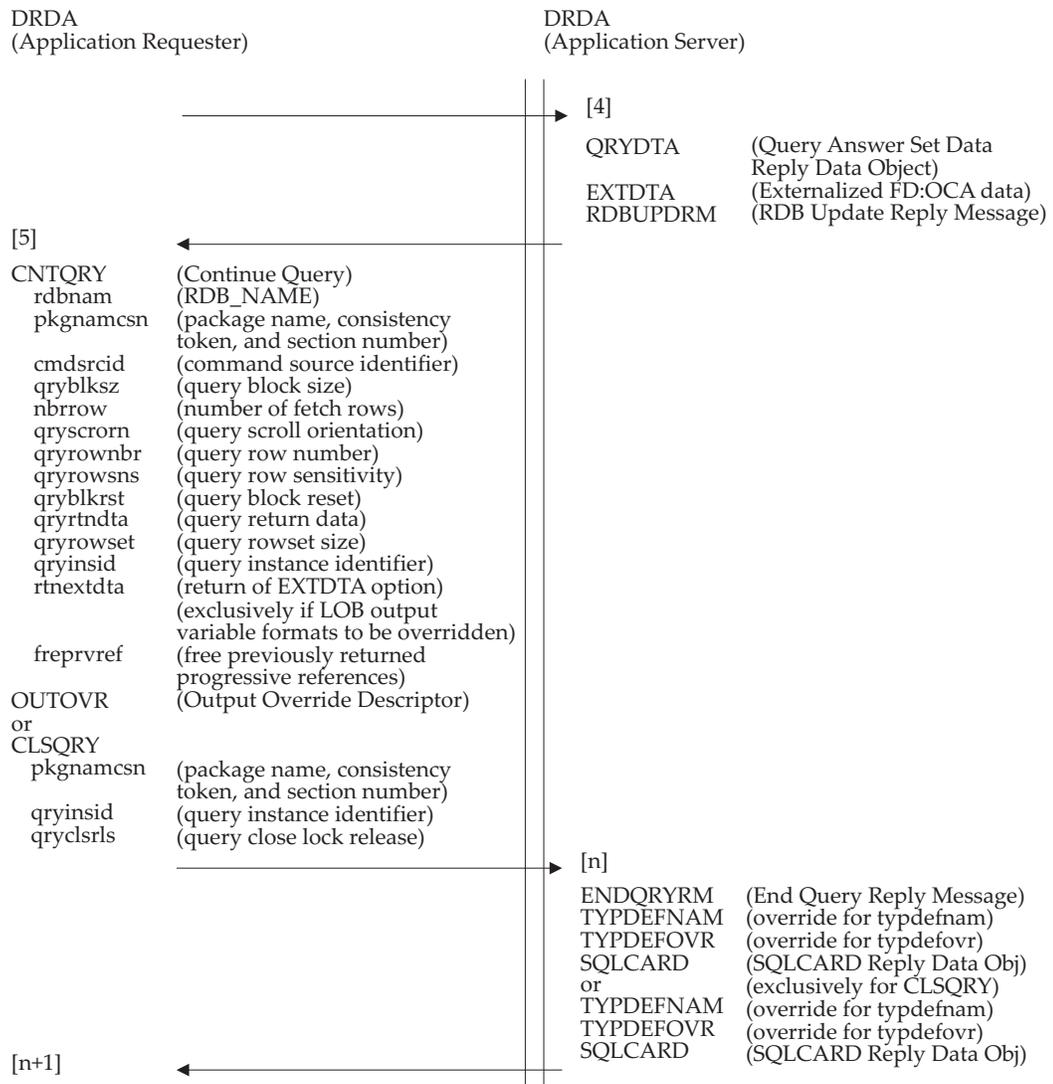


Figure 4-25 Limited Block Protocol Query Processing (FD:OCA Generalized String Data, rtnextall) (Part 2)

**Limited Block Protocol (rtnextdta=rtnextall)**

This discussion is based on [Figure 4-24](#) (on page 165), [Figure 4-25](#) (on page 166), and the discussion in [Section 4.4.6.2](#) (on page 157).

1. Refer to [Section 4.4.6.2](#) (on page 157), Step 1.
2. Refer to [Section 4.4.6.2](#) (on page 157), Step 2.

If DYNDTAFMT is set to FALSE and the answer set contains FD:OCA Generalized String data, no query data nor externalized data is returned in response to the OPNQRY.

If DYNDTAFMT is set to TRUE and the answer set contains LOB data, then no QRYDTA nor EXTDTA is returned in response to the OPNQRY unless OUTOVROPT is set to OUTOVRNON. This allows an OUTOVR to be specified on subsequent CNTQRY requests.

If DYNDTAFMT is set to TRUE and the answer set does not contain LOB data, then the QRYDTA and/or EXTDTA including other FD:OCA Generalized String data is returned in response to the OPNQRY. An OUTOVR cannot be specified on subsequent CNTQRY requests.

3. Refer to [Section 4.4.6.2](#) (on page 157), 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 the one determined by the application server using Data Format (DF Rules), 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 to the application server, 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 and Data Format (DF Rules). For other non-LOB FD:OCA Generalized Strings, the data is returned according to the Data Format (DF Rules); see [Section 7.8](#) for more details.

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 157), 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 157), Step 4.

If any FD:OCA Generalized String columns are in the row being retrieved, data will be returned as follows. All LOB locators are returned in the QRYDTA object. Each FD:OCA Generalized String column is returned according to the Data Format (DF Rules); see [Section 7.8](#) for more details. If more than one EXTDTA is returned, they are returned in the order that their corresponding FD:OCA Generalized String headers 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 the complete rows contained in the QRYDTA object. Thus, in the case of exact blocking, 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 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 does support the return of extra query blocks, however, it must adhere to certain rules. For example, if exact blocking rules are used, the last extra query block sent must contain the end of at least one complete row.

If the application server returns extra query blocks when there is externalized data 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.

The QRYDTA reply chain can optionally be followed by an RDBUPDRM reply message as per the Update Control (UP Rules) in [Section 7.19](#) (on page 467).

5. Refer to [Section 4.4.6.2](#) (on page 157), Step 5.

The application requester receives the QRYDTA block. The first row data received in the QRYDTA is mapped to the application's host variables. FD:OCA Generalized Strings are obtained according to the Data Format (DF Rules); see [Section 7.8](#) for more details. EXTDTAs are flown 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 Generalized String data from the QRYDTA block and, depending on the mode in the header, obtaining the FD:OCA Generalized String data from the QRYDTA or the associated EXTDTAs. 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.

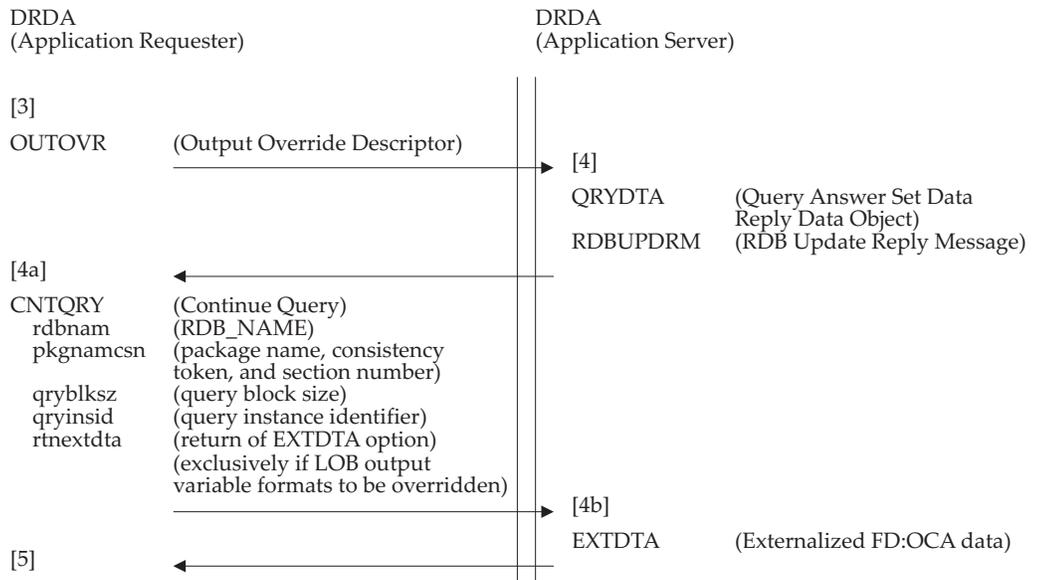
- n* Refer to [Section 4.4.6.2](#) (on page 157), Step *n*.

If an FD:OCA Generalized String 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 157), Step *n*+1.

**Limited Block Protocol (rtnextdta=rtnextrow)**

This discussion is based on Figure 4-24 (on page 165), Figure 4-25 (on page 166), and the discussion in Section 4.4.6.3 (on page 164). Additional commands required for this case are included in Figure 4-26 (on page 169).



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

**Figure 4-26** Limited Block Protocol Query Processing (Externalized Data, rtnextrow)

1. Refer to Limited Block Protocol (rtnextdta=rtnextall) (on page 167), Step 1.
2. Refer to Limited Block Protocol (rtnextdta=rtnextall) (on page 167), Step 2.
3. Refer to Limited Block Protocol (rtnextdta=rtnextall) (on page 167), Step 3.
4. Refer to Limited Block Protocol (rtnextdta=rtnextall) (on page 167), 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. The QRYDTA reply chain can optionally be followed by an RDBUPDRM reply message as per the Update Control (UP Rules) in Section 7.19 (on page 467).

- 4a FD:OCA Generalized String column is trivial if one of the following is true:
  1. It is a nullable column and it is null.
  2. The length of the FD:OCA Generalized String header is zero.

If the base data row is complete and all of its FD:OCA Generalized String columns are trivial or have a mode byte value of X'01' or X'03' in the FD:OCA Generalized String headers, then there are no pending EXTDTA objects for this row and the application requester returns to the application with the fetched base data.

If the base data row is complete and all of its FD:OCA Generalized String columns have a mode byte value of X'02' in the FD:OCA Generalized String headers, the application requester sends a CNTQRY command to the application server to obtain the associated EXTDTA objects for the row.

If a partial base data row is encountered (when exact blocking rules are used) or if there is no more data in the QRYDTA object, this step reverts to Step 3 in [Section 4.4.6.2](#) (on page 157), where the CNTQRY command is sent to obtain additional base data.

- 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 FD:OCA Generalized String columns pending for that base row.

If there are no more pending EXTDTA objects to be sent for previously sent rows, this step reverts to Step 4 in [Section 4.4.6.2](#) (on page 157), where the CNTQRY command causes the application server to obtain additional base data, including completing a partial row in the case of exact blocking.

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 157), Step 5.  
*n* Refer to [Section 4.4.6.2](#) (on page 157), Step *n*.  
*n*+1 Refer to [Section 4.4.6.2](#) (on page 157), 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](#) describes the commands and replies that flow in most instances. [Section 4.4.7.2](#) 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.

The EXCSQLSET command requires package name and consistency token parameters, but no section number parameter, as it is not bound into a package. Support for the SET CURRENT PACKAGE PATH statement is contingent on support of the EXCSQLSET command, as this value is propagated from a requester to a database server (possibly through intermediate servers) using the EXCSQLSET command.

##### 4.4.7.1 Executing Ordinary Bound SQL Statements

[Figure 4-27](#) 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 177).

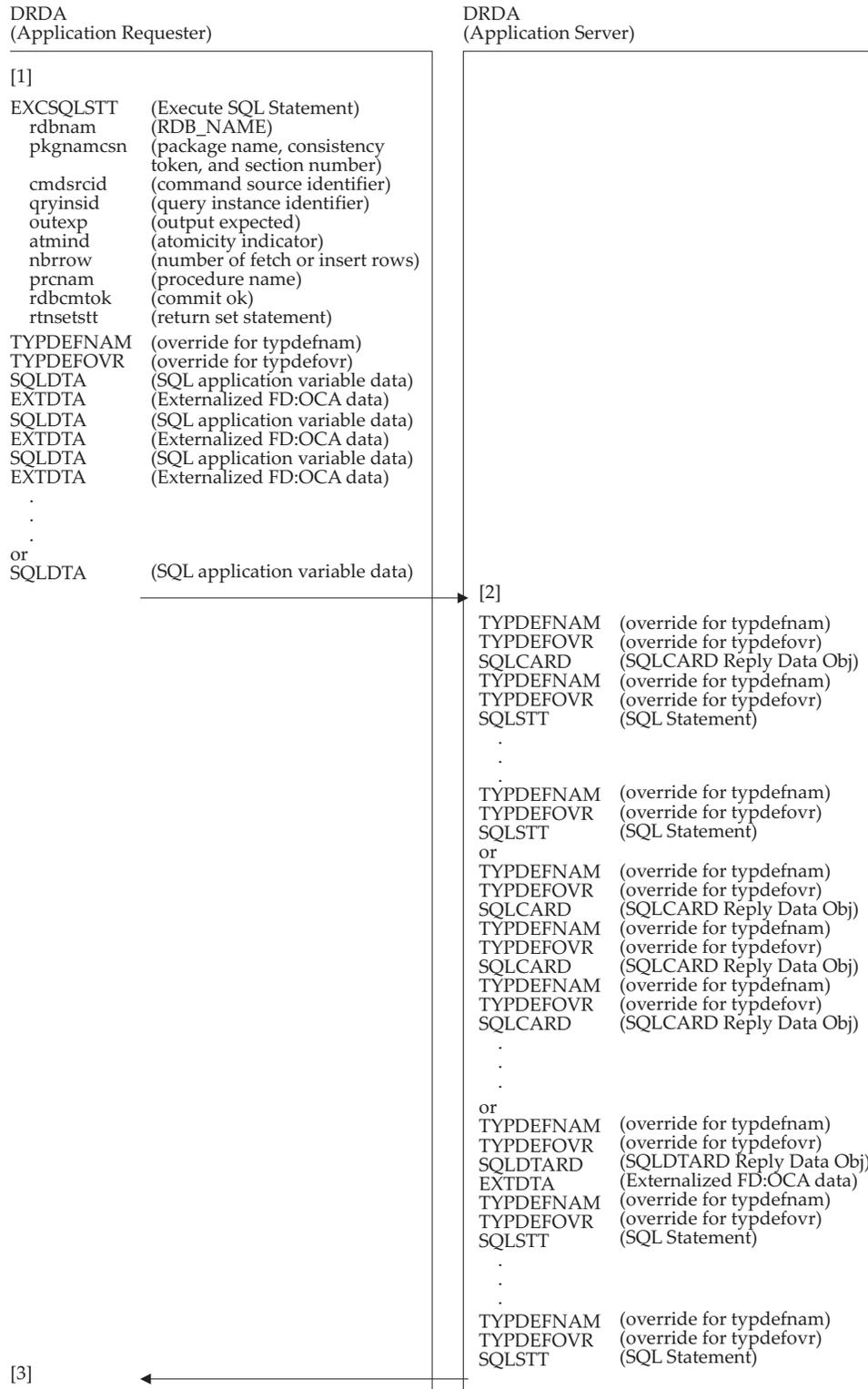


Figure 4-27 Executing a Bound SQL Statement

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 89)), prebound SQL statements referenced in a package in a remote relational database can be executed. (See [Section 4.4.3](#) 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 optional *cmdsrcid* parameter uniquely identifies the source of the command. If the SQL statement being executed is a positioned delete/update, then the *qryinsid* parameter must also be specified in order to indicate the instance of the query in use, unless only a single query instance exists for the section. The application requester also indicates in the *outexp* parameter whether or not it expects output to be returned within an SQLDARD reply data object as a result of the execution of the SQL statement. The optional *rtinsetstt* parameter specifies whether the server must return one or more SQL SET statements for any special registers whose settings have been changed on the current connection, if the execution of the command causes any special register setting to be updated. For a multi-row input operation, no output is allowed, and the *nbrrow* parameter will be mandatory. The optional *atmind* parameter indicates whether the multi-row input operation is atomic or non-atomic, and the *nbrrow* parameter indicates the number of rows for the input operation. The *nbrrow* parameter is allowed to have a value of 1. The optional *rdcbmtok* 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. If this is a multi-row input operation, and there is no data that is to flow in associated EXTDTA objects, then only one SQLDTA command data object is required based on the SQLDTAMRW multi-row input RLO descriptor. Otherwise, each input data row is flowed in a separate SQLDTA command data object, in which case only the first SQLDTA is allowed to be preceded by a TYPDEFNAM and/or TYPDEFOVR 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 command 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 FD:OCA Generalized String 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 an FD:OCA Generalized String, an FD:OCA Generalized String header is placed in SQLDTA and the corresponding value bytes are flowed in an EXTDTA following the SQLDTA. The FD:OCA Generalized String is formatted according to Rule DF7; see [Section 7.8](#) for more details. The EXTDTAs flow in the order that the FD:OCA Generalized String headers occur in the associated SQLDTA.

If this is a multi-row input operation, the SQLDTA command data object for an input data row is followed immediately by any EXTDTA data object(s) containing the externalized data value(s) for that row. If the application requester encounters an error in setting up a particular row in an SQLDTA or EXTDTA OBJDSS within the multi-row input chain, the row can be marked as null using the null indicator for the SQLDTAGRP FD:OCA object.

While typically this null indicator will be used to indicate that a multi-row input row follows, in the case of a null row, the value of the null indicator will indicate to the server to either skip over this row, or to issue an error instead.

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.

The optional *rtnsqlda* parameter controls whether describe information is returned or not; if not specified, no describe information is returned. The optional *typsqlda* parameter can request light, standard, or extended descriptive information to be returned. The *typsqlda* is not a required parameter because the FDODSC object provides enough row-level information needed to parse the SQLDTARD and return the parameter values to the application. This describe information provides descriptive information about output parameters. The descriptive information is returned in the SQLDARD as reply data.

2. The application server receives the EXCSQLSTT command. If the statement call is a CALL statement, then CALL-specific instance variables, such as *qryblksz*, *maxblkext*, *maxrslcnt*, or *rslsetflg*, are applied to the execution of the statement. Otherwise, CALL-specific instance variables are ignored. The application server 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. If this is an atomic multi-row input operation, there is only one reply for all the input data rows which is normally an SQLCARD reply data object. If the multi-row input operation is non-atomic, there is one reply for each input data row. The SQLDTARD reply data object is not applicable for a multi-row input operation as no output is allowed.

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 any special register has been updated during execution of this command, as per the setting of the optional *rtsetsstt* parameter, the server may return one or more SQLSTT reply data objects, each containing an SQL SET statement for a special register whose setting has been changed on the current connection.

If this is a multi-row input operation, and the value of the *nbrrow* parameter does not match the number of input data rows (including any null rows) in the SQLDTA reply data object, the application server returns PRCCNVVM with a *prccnvcd* value of X'1E' to the application requester. For any null row within the multi-row chain, the server will skip over that row if its null indicator so indicates. However, if instead the null indicator indicates an error should be issued over the null row, an error SQLCA containing SQLSTATE 22527 should be returned by the server. In particular, for an atomic multi-row input operation, such an error terminates processing of the multi-row input request, and any changes that have resulted from this request will be undone.

If the executed SQL statement is either a COMMIT or ROLLBACK, see [Section 4.4.15.1](#) and [Section 4.4.15.2](#) 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 an FD:OCA Generalized String, an FD:OCA Generalized String header is placed in the SQLDTARD and the corresponding value bytes are flowed in an EXTDTA following the SQLDTARD. The FD:OCA Generalized String is formatted according to Rule DF7; see [Section 7.8](#) for more details. The EXTDTAs flow in the order that the FD:OCA Generalized String headers occur in the associated SQLDTARD.

If an OUTOVR object is received, it will be used to format the LOB 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.

Depending on the value of the *rtnsqlda* and *typsqlda* parameters on the EXCSQLSTT command, an optional SQLDARD is generated to provide descriptive information about the statement. The SQLDARD is generated only if the SQL statement is successful. The level of describe information returned is identified by the value of the *typsqlda* parameter. The SQLDARD is returned before the SQLDTARD; otherwise, the *rtnsqlda* parameter is ignored if there is no SQLDTARD returned.

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. The application requester also caches all SQL SET statements that may have been returned from the server in SQLSTT reply data objects so that they can be used later to restore the execution environment when the connection is reestablished to the database at either the original location or an alternate failover location in case of a communications failure.

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.

If this is a non-atomic multi-row input operation, the application requester may be able to report to the application the outcome for each input data row based on the SQLCARD reply data object which is returned for each row or it may only return one SQLCA to the application for all the rows. Details of how the application requester chooses to address this issue are implementation-specific and may be influenced by the API used by the application.

#### 4.4.7.2 Invoking a Stored Procedure that Returns Result Sets

Figure 4-28 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 *outovropt*, 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 7.22 (on page 473).

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. This example briefly discusses some topics that relate to scrollable result sets. For more information, refer to Section 4.4.6.1 (on page 148), and Section 4.4.6.2 (on page 157), as well as to the Scrollable Cursor Overview given in Appendix B (on page 707).

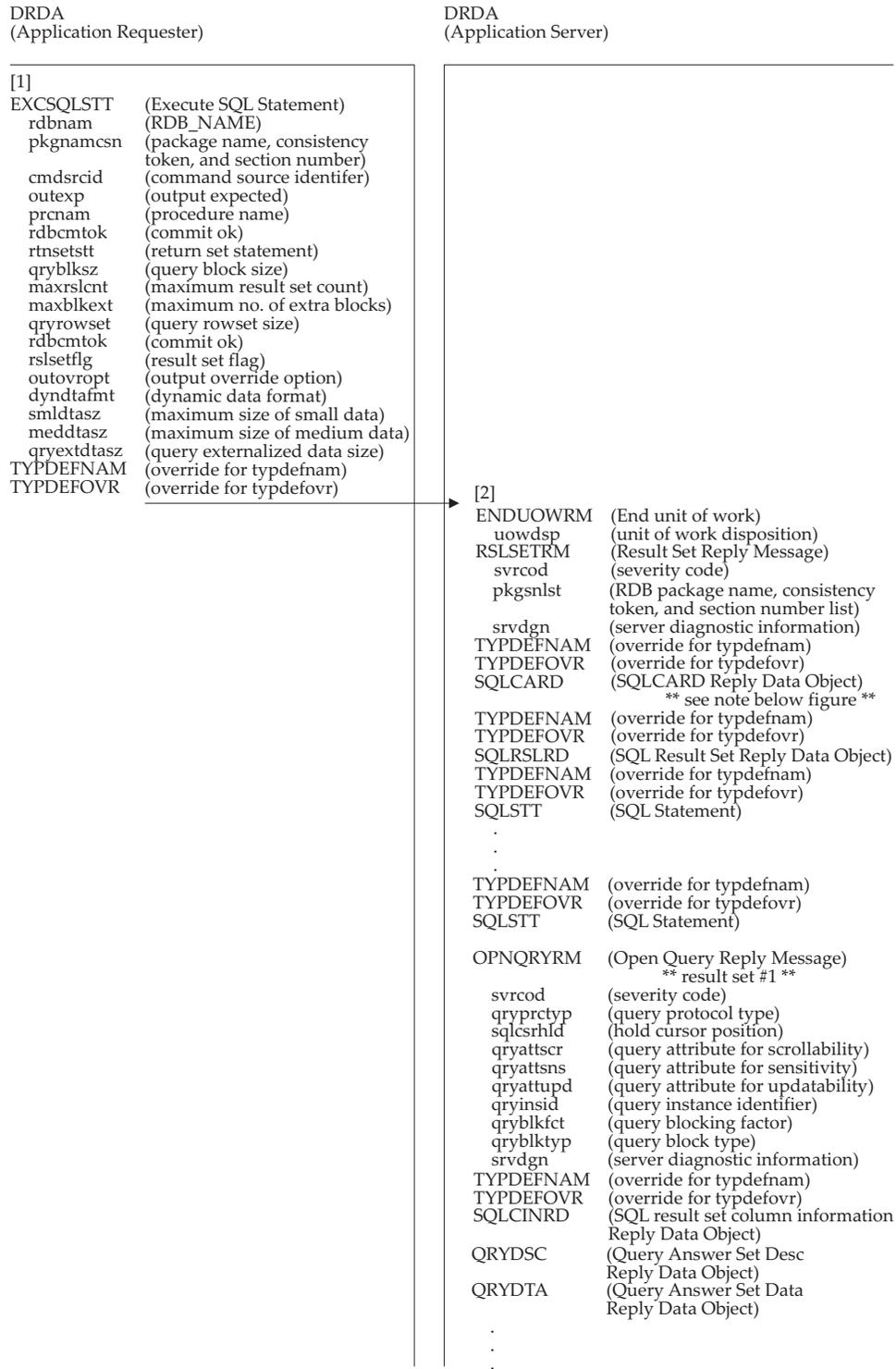


Figure 4-28 Executing a Stored Procedure (Part 1)

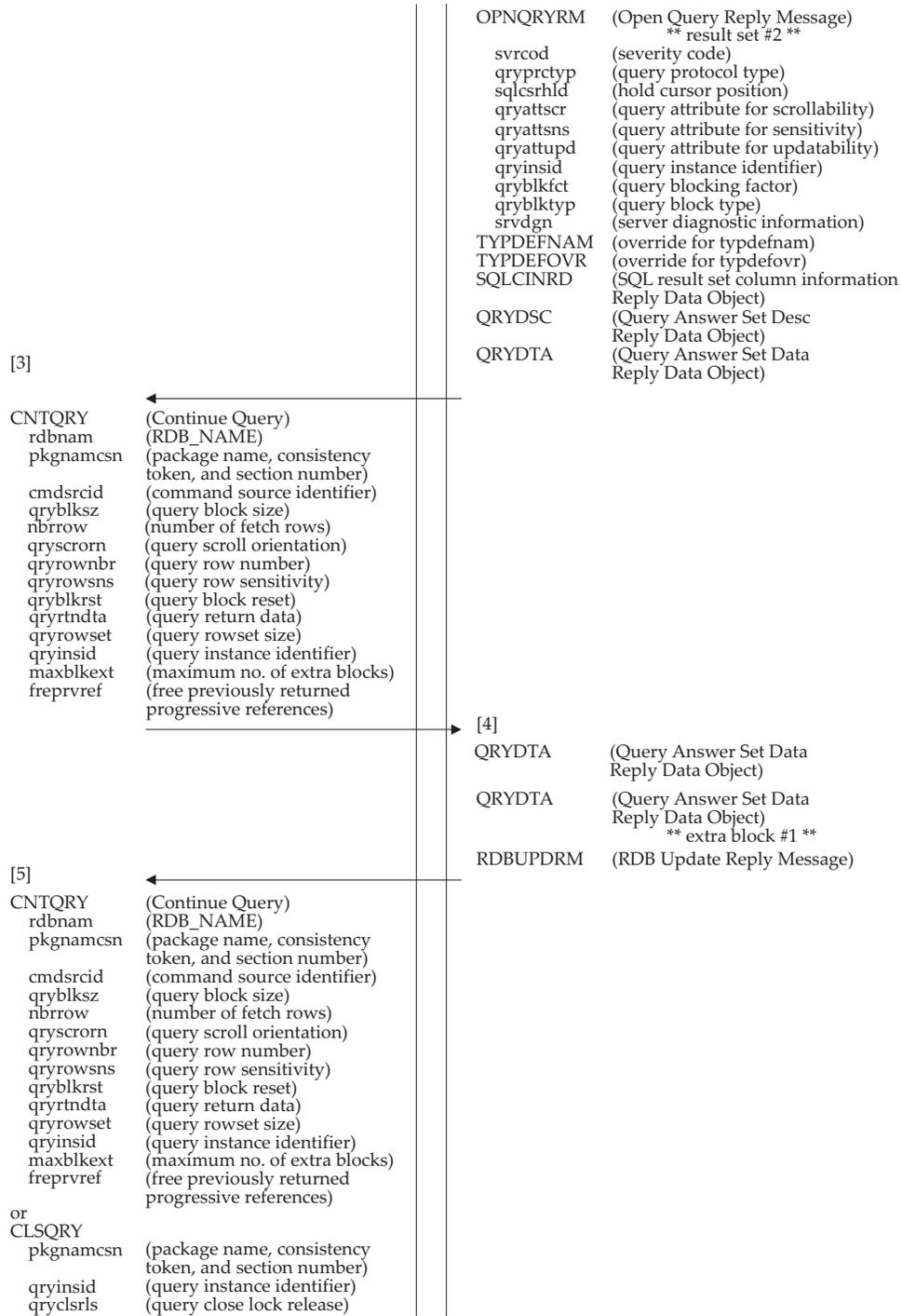
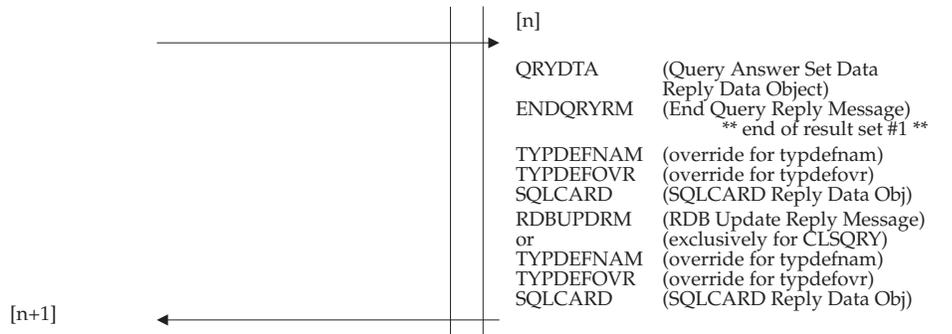


Figure 4-29 Executing a Stored Procedure (Part 2)



**Figure 4-30** Executing a Stored Procedure (Part 3)

**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 4-28](#) and [Figure 4-29](#) assume that there are no FD:OCA Generalized Strings in any of the result sets, the following discussion indicates where FD:OCA Generalized String-related processing occurs. Refer to [Section 4.4.6](#) for additional discussions of FD:OCA Generalized String-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 89)), prebound SQL statements referenced in a package in a remote relational database can be executed. (See [Section 4.4.3](#) 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 optional *cmdsrcid* parameter uniquely identifies the source of the command, and in this case one or more result sets. The optional *rtmsetstt* parameter specifies whether the server must return one or more SQL SET statements for any special registers whose settings have been changed on the current connection, if the execution of the command causes any special register setting to be updated. 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<sup>41</sup> *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 10M bytes. If not, the application server returns the VALNSPRM reply message and the application server does not execute the command.

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

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 *rdbcmntok* 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 *qryrowset* parameter on the EXCSQLSTT applies to a result set returned by the stored procedure only if the result set is a non-scrollable, non-rowset cursor and conforms to the limited block query protocol or if the result set is scrollable and not sensitive dynamic. The *qryrowset* parameter specifies whether the indicated number of single-row fetches are to be attempted in order to return a DRDA rowset with the OPNQRYRM for each such result set for which the *qryrowset* parameter is applicable. The same *qryrowset* value applies to all such result sets returned. If there are no result sets to which the parameter applies, including when all scrollable or rowset result sets are reverted to non-scrolling, non-rowset result sets, the *qryrowset* parameter on the EXCSQLSTT is ignored. The application requester sends the command and command data to the application server.

The optional *dyndtafmt* parameter with the value TRUE (X'F1') requests that the application server returns FD:OCA Generalized String according to Rule DF5; see [Section 7.8](#) for more details. The optional *smldtasz* parameter specifies the maximum size of the data in bytes to be flown in Mode X'01' of Dynamic Data Format. The optional *meddtasz* parameter specifies the maximum size of the data in bytes to be flown in Mode X'02' of Dynamic Data Format. *dyndtafmt*, *smldtasz*, and *meddtasz* are parameters that apply to result sets and not to output parameters.

The optional *rtnsqlda* parameter controls whether describe information is returned or not; if not specified, no describe information is returned. The optional *typsqlda* parameter can request light, standard, or extended column descriptive information to be returned. The *typsqlda* is not a required parameter because the FDODSC object provides enough row-level information needed to parse the SQLDTARD and return the parameter values to the application. This describe information provides descriptive information about output parameters. The descriptive information is returned in the SQLDARD as reply data. The SQLDARD is returned only if the CALL is successful and there are parameters.

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 the application server may return per result set to two. The *qryblkz* 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 *rdcbmtok* was specified as TRUE in the EXCSQLSTT command, the application server initiates commit processing. When commit processing completes successfully, the application server builds the transaction component of the reply consisting of the ENDUOWRM<sup>42</sup> with *uowdsp* set to committed. 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.<sup>43</sup> 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. As per the setting of the optional *rtinsetstt* parameter, if any special register has been updated during execution of the EXCSQLSTT command, following the SQLRSLRD reply data object, the summary component may contain one or more SQLSTT reply data objects, each containing an SQL SET statement for a special register whose setting has been changed on the current connection.

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 (possibly with an optional SQLCARD), an SQLCINRD, and the FD:OCA description of the data (QRYDSC). The application server may return answer set data in a query block consisting of a QRYDTA reply data object as long as it is permitted by the query protocol rules. Additional blocks of answer set data may also be chained to the block, 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. The server may optionally return an RDBUPDRM reply message at the end of the reply chain (as per the UP Rules in Section 7.19 (on page 467)).

If the application server returns FD:OCA Generalized String according to Rule DF5, the *dyndtafmt* parameter is returned with the value TRUE (X'F1').

For a non-scrollable, non-rowset result set, the server uses the *qryrowset* parameter specified on the EXCSQLSTT to return a DRDA rowset in the QRYDTA for the cursor. If any result

---

42. The reply messages that can be returned as part of the transaction component include ENDUOWRM, RDBUPDRM, or CMMRQSRM. If RDBUPDRM is returned (as per the UP Rules in Section 7.19 (on page 467)), it may be followed by either ENDUOWRM or CMMRQSRM.

43. 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.

sets returned by the stored procedure are scrollable or rowset result sets, then the server first checks that the requester supports scrollable cursors or rowsets. If the requester is not at SQLAM Level 7 or higher, then the server acts according to whether it is the target server or an intermediate server. The target data server reverts all scrollable or rowset result sets to non-scrolling or non-rowset result sets while an intermediate server fails the stored procedure call with an SQLSTATE of 560B3. When processing a scrollable cursor, the server then validates that all scrollable result sets are positioned before the first row of the corresponding result table. If any result sets are invalidly positioned, the stored procedure call is failed with an SQLSTATE of 560B1. If each non-scrollable, non-rowset result set is validly positioned, the application server uses the *qryrowset* parameter to return a DRDA rowset in the QRYDTA for each such query. If the *qryrowset* parameter is not provided on the EXCSQLSTT, the application server either returns no rows (for Fixed Row Protocol, including for rowset cursors) or returns an implicit DRDA rowset (for Limited Block Protocol) according to Rule QP4 (see [Section 7.22.3](#) (on page 484)).

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 data for the cursor until the application at the application requester issues a FETCH request, unless the OUTOVRPT is set to OUTOVRNON, DYNDTAFMT is TRUE, and QRYPRCTYP is LMTBLKPRC. So, for each such cursor returned by the stored procedure, the application server returns:

- OPNQRYRM
- SQLCINRD
- QRYDSC

If any cursors in the stored procedure will result in any non-LOB FD:OCA Generalized String data being returned in the answer set, then the application server does not return any data for the cursor until the application requester issues a FETCH request, unless DYNDTAFMT is TRUE and QRYPRCTYP is LMTBLKPRC.

In this sample flow, the response to the EXCSQLSTT command consists of DSSs comprising the ENDUOWRM reply message in the transaction component, the summary component, and two result set components. The summary component consists of the DSSs starting with the RSLSETRM through the SQLSTT reply data objects. The result set component for the first result set consists of the OPNQRYRM reply message, the SQLCINRD reply data object, the QRYDSC reply data object, and three QRYDTA reply data objects, each of which is a query block. The result set component for the second result set consists of the OPNQRYRM reply message, the SQLCINRD reply data object, the QRYDSC reply data object, a QRYDTA reply data object which is a query block, an ENDQRYRM reply message, and an SQLCARD reply data object. In this example, each OPNQRYRM reply message has a *qryprctyp* parameter value of LMTBLKPRC. Also, the reply chain ends with an RDBUPDRM reply message (as per the UP Rules in [Section 7.19](#) (on page 467)). The application server sends the response to the application requester.

For each OPNQRYRM reply message for result sets that do not have the rowset attribute, the value of the *qryprctyp* parameter is LMTBLKPRC in this example. For rowset result sets, the value of the *qryprctyp* parameter is FIXROWPRC. Depending on whether the cursor is scrollable and whether the cursor returns LOB data, the *qryprctyp* can also be FIXROWPRC instead. This flow is not shown in this example. The server sends the response to the requester.

Depending on the value of the *rtnsqlda*, *rslsetflg*, and *typsqlda* parameters on the EXCSQLSTT command, an optional SQLDARD and SQLCINRD, if requested, is generated to provide descriptive information about the stored procedure parameters and result sets. The SQLDARD is generated only if the CALL is successful and there are in/out or output

parameters. The level of describe information returned is identified by the value of the *typsqlda* parameter. The SQLDARD is returned before the SQLDTARD; otherwise, the *rtnsqlda* parameter is ignored if there is no SQLDTARD returned. The *rslsetflg* controls the level of describe information returned for each result set.

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 4.4.15.2](#) 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 also caches all SQL SET statements that may have been returned from the server in SQLSTT reply data objects so that they can be used later to restore the execution environment when the connection is reestablished to the database at either the original location or an alternate failover location in case of a communications failure.

The application requester receives the DSSs in each result set component from the application server. The application requester associates each DSS with the *pkgnamcsn*, *qryinsid*, locator value, and name of its result set and then stores the description and answer set data associated with each result set for a subsequent FETCH by the application at the application requester. Depending on the QRYBLKTYP selected by the server for the result set, the answer set data may be returned in flexible query blocks or in exact query blocks. If flexible query blocks are returned, then only complete base data rows are included in the query blocks. If exact query blocks are returned, the last query block may end with a partial row.

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 application requester 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 the one determined by the application server according to the last OUTOVR and the Data Format (DF Rules); see

[Section 7.8](#) for more details.

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, along with the optional *cmdsrcid* parameter to uniquely identify the result set. The CNTQRY command also specifies the mandatory *qryinsid* value for the rest set as previously. 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.

The optional *freprvref* parameter specifies whether all the previously returned progressive references in a complete row should be freed upon completion of this command.

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, the optional *cmdsrcid* parameter that uniquely identifies the source of the result set, and also the *qryinsid* parameter that uniquely identifies the instance of the result set. If one or more exact query blocks were previously returned and the last block ended with a partial row, the application server places the remainder of the partial row in the reply chain, consuming one or more exact query blocks as needed. If flexible query blocks were previously returned or if this is the first CNTQRY, the application server retrieves a data row from the answer set and places it in the QRYDTA reply data object along with an SQLCA. The block containing the 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 corresponding data in the format given by the override descriptors.

If the FD:OCA Generalized String is to be flown as externalized data, the FD:OCA Generalized String header flows in the QRYDTA object, and the data flows in the associated EXTDTA object.

As per the Update Control (UP Rules) in [Section 7.19](#) (on page 467), an optional RDBUPDRM reply message may be flowed at the end of this reply chain.

In this sample flow, the response to the CNTQRY command consists of two blocks, followed by an RDBUPDRM reply message. 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 handles the received row data according to whether exact or flexible query blocks are received. If exact query blocks are received and a partial row was received in reply to the previous command, the application requester must assemble the row data from the previous command with the remaining row data in the current reply chain before passing the row to the application. If flexible blocking is in effect or if exact blocking is in effect but there is no pending partial row, the application requester passes the received row data to the application.

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, along with the optional *cmdsrcid* parameter to uniquely identify the result set. The CNTQRY command also specifies the mandatory *qryinsid* value for the result set as previously. 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.

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

- 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, the optional *cmdsrcid* parameter that uniquely identifies the source of the result set, and also the *qryinsid* parameter that uniquely identifies the instance of the result set. As described in Step 4, it places the remaining portion of a partial row or retrieves the next desired data row from the answer set and places it in the QRYDTA reply data object along with an SQLCA. The application server continues to retrieve additional rows of the answer set and to place them in the QRYDTA reply data object until it retrieves 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 query block is complete. The application server may generate an ENDQRYRM reply message and an SQLCARD reply data object and chain them to the query block. In such a case, the server closes the query. Otherwise, the server does not send the ENDQRYRM and does not close the query. This reply chain can optionally be followed by an RDBUPDRM reply message as per the Update Control (UP Rules) in [Section 7.19](#) (on page 467).

For non-scrollable queries, 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 may either return the query block and leave the query open, or close the query implicitly. In the case of a scrollable cursor, the query must stay open.

If the cursor is scrollable, or if the server has determined that the query cannot be closed implicitly based on some other cursor properties, the server does not close the query until a CLSQRY command is received from the application requester, or until the transaction is rolled back.

The server does not close the cursor implicitly, even though the end of the query data is reached, if one of the following conditions is true:

1. There are any previously returned progressive references that belong to a complete row and are not freed, and FREPRVREF=TRUE is not specified with the CNTQRY command.
2. There are any previously returned progressive references that belong to an incomplete row.
3. There are any new progressive references to be returned upon completion of the CNTQRY command.

Otherwise, the server may choose to close the non-scrollable query implicitly at this time. For non-scrollable result sets, after it has placed the last row of the answer set (which completes the last QRYDTA reply data object) in the query block, then the query block is complete. The application server can close the query implicitly by placing an ENDQRYRM reply message and the associated SQLCARD reply data object in the reply chain after the

last query block. If Rule CH5 (see [Section 7.22.1.3](#) (on page 478)) prohibits the server from closing the query implicitly, then the server does not send the ENDQRYRM and its associated SQLCARD to the application requester with this reply chain. They will be sent as the only responses to the next CNTQRY command.

If the server sends the ENDQRYRM reply message and the SQLCARD reply data object to the application requester, then the query is terminated. The server closes the query and will no longer accept CNTQRY commands for this cursor until it is reopened.

For scrollable cursors and other cursors that the server does not close implicitly, reaching the end of the query data (SQLSTATE 02000) does not terminate the cursor, so the ENDQRYRM reply message should not be returned and the query is not closed until the application requester sends a CLSQRY command. In response to the CLSQRY command, the server sends back an SQLCARD reply data object.

*n*+1 When the application requester receives the QRYDTA reply data object, it passes the row data to the application as described in Step 5.

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 message, it knows that the application server has either processed the last row of answer data (for a non-scrollable cursor) or that a terminating error has occurred. It knows the application server has closed the query (for a non-scrollable cursor) or that the cursor has been terminated and is in a not-opened state, so the application requester will not send any additional CNTQRY commands to the application server.

If the application requester has previously sent out a CLSQRY command to explicitly close the open query, the reply will consist of an SQLCARD reply data object.

#### 4.4.7.3 Executing Chained Ordinary Bound SQL Statements as an Atomic Operation

Execute SQL Statement (EXCSQLSTT) commands can be batched as a single atomic operation. The response consists of a chain of replies, one for each EXCSQLSTT in the command chain. Because the operation is atomic, all replies except the one corresponding to the last EXCSQLSTT command processed in the chain must indicate success. [Figure 4-31](#) shows the flows involved.

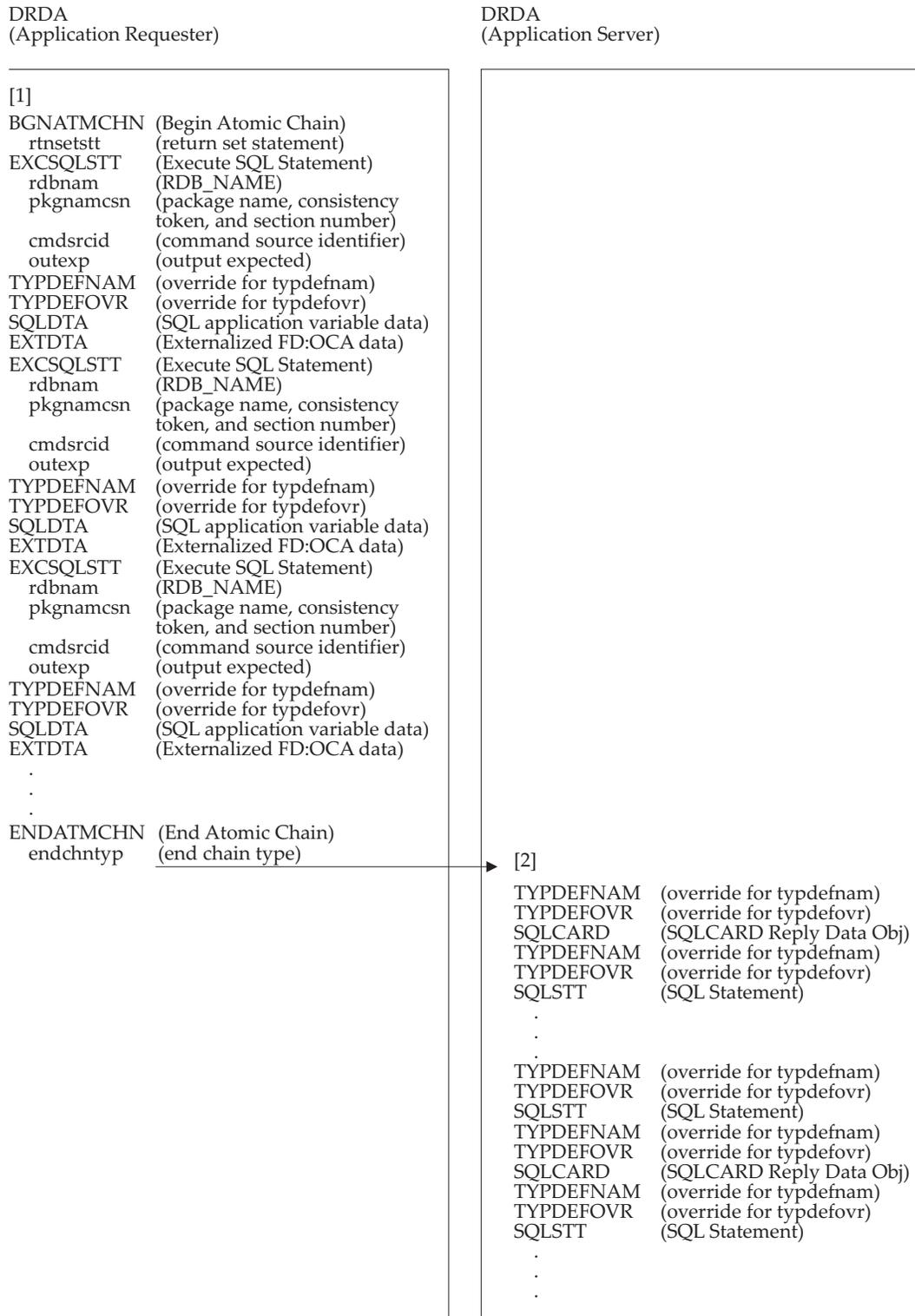


Figure 4-31 Executing Bound SQL Statements as an Atomic Operation (Part 1)

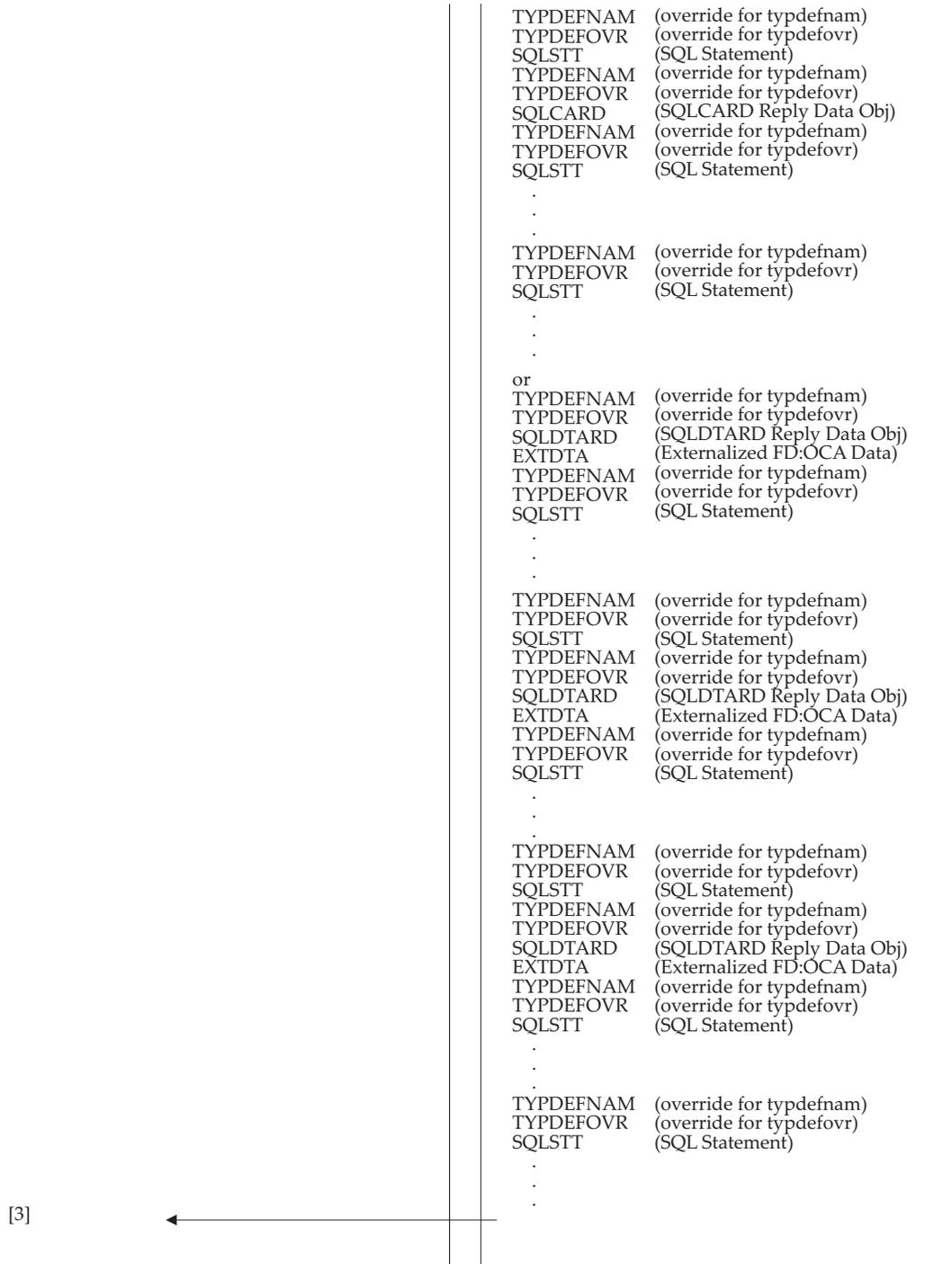


Figure 4-32 Executing Bound SQL Statements as an Atomic Operation (Part 2)

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. The application requester starts an atomic chain with a Begin Atomic Chain (BGNATMCHN) command. The application requester then creates a chain of Execute SQL Statement (EXCSQLSTT) commands on behalf of the application. Refer to [Section 4.4.7.1](#) for details of the EXCSQLSTT command. The following restrictions are imposed on each of the EXCSQLSTT commands in the atomic chain:
  1. The EXCSQLSTT command cannot invoke a commit or rollback.
  2. The EXCSQLSTT command cannot be for a CALL statement for a stored procedure.

Note that an EXCSQLSTT command in the chain may be for a select statement, in which case its reply can be an SQLDTARD instead of an SQLCARD.

In addition, it is highly recommended although not mandatory that the application requester make use of the default package name and consistency token on each EXCSQLSTT command in the chain as per Rule CU14 in [Section 7.6](#) whenever possible.

For each EXCSQLSTT in the atomic chain, there is no change regarding how input application variable data (both non-LOB and LOB) is flowed in the SQLDTA command data object and the optional EXTDTA data object.

It is permissible to have an EXCSQLSTT command representing a multi-row input operation in the atomic chain. Since the chain is atomic, the multi-row input operation contained therein must also be atomic.

The application requester terminates the atomic chain with an End Atomic Chain (ENDATMCHN) command.

2. The BGNATMCHN command informs the application server that this is the start of an atomic EXCSQLSTT chain. There is no reply required for this command if it is processed successfully. The application server then receives and processes each of the EXCSQLSTT commands in the atomic chain in sequence and creates an SQLCARD reply data object or SQLDTARD reply data object for each command. Any LOB data value in the output which needs to be externalized will be flowed back in an EXTDTA data object following its SQLDTARD reply data object. Each 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. For all EXCSQLSTT commands in the atomic chain, if the statement executes successfully (with or without warnings), an SQLCARD or SQLDTARD reply data object is returned. As per the setting of the optional *rtsetsst* parameter as specified on the BGNATMCHN command, if any special register has been updated during execution of an EXCSQLSTT command within the atomic chain, its reply must also contain one or more SQLSTT reply data objects, each containing an SQL SET statement for a special register whose setting has been changed on the current connection. Processing of the atomic chain stops immediately upon the first error condition, or when the ENDATMCHN command has been processed successfully. If an error has occurred, all changes made as a result of previous successful statement executions earlier in the atomic chain are undone.

The ENDATMCHN command is only processed if all prior BGNATMCHN and EXCSQLSTT commands in the atomic chain have been processed successfully. This command indicates to the application server the end of the atomic chain. The optional *endchntyp* parameter enables the application requester to inform the application server either to terminate the chain normally or to abort the chain. If the ENDATMCHN command is processed successfully, the application server returns an SQLCARD reply data object

indicating success for the entire atomic chain.

3. For a normal completion, the application requester returns to the application with the successful indication which applies to all of the EXCSQLSTT commands in the chain. The application requester may also choose to return to the application any warning conditions set in the SQLCAGRP for individual EXCSQLSTT commands in the chain. The application requester also returns data in each of the SQLDTARD reply data objects in the reply chain, and if applicable, its associated EXTDTA data object(s) to the application. The application requester also caches all SQL SET statements that may have been returned from the server in SQLSTT reply data objects so that they can be used later to restore the execution environment when the connection is reestablished to the database at either the original location or an alternate failover location in case of a communications failure.

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

If the last SQLCARD reply data object that the application server returned to the application requester indicates that the atomic EXCSQLSTT chain was not successful, the application requester returns an exception to the application that is attempting to execute the chain of SQL statements.

4.4.7.4 Executing Bound SQL Statement with Array Input

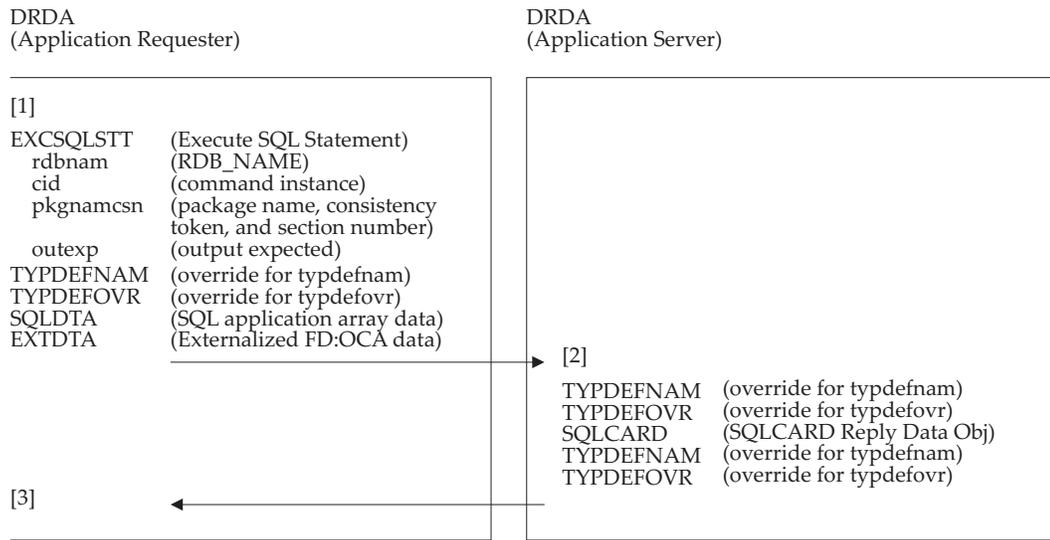


Figure 4-33 Executing an Array Input SQL Statement

1. After the requester and the server have established proper connection (described in Figure 4-2 (on page 89)), prebound SQL statements referenced in a package in a remote relational database can be executed. (See Section 4.4.3 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. In order to flow the variable arrays using the SQLDTA object, the server is required to return the extended diagnostics in the reply data. Extended diagnostics are described by the SQLDIAGGRP early descriptor and allows the server to return multiple warning or error conditions for a single request in the SQLCARD.

The requester that is acting as the agent for the application performing the execute of an array input SQL statement function creates the Execute SQL Statement (EXCSQLSTT) command or an Open Query (OPNQRY) command (not illustrated by this example) by providing the correct command instance, package name, consistency token, and section number in the *pkgnamcsn* parameter. For this example, 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 SQLDTA contains the description and the data for each input array. It consists of an FDOEXT data object, an FDODSC descriptor object, an FDODTA data object, and an FDOOFF data object. First, the FDOEXT data object is built. The FDOEXT data object contains the extent specification for each field in the SQLDTA. The extent specification defines the number of times the field is repeated. Second, the FDODSC descriptor is built. The FDODSC descriptor object describes each repeatable field. Each field describes an array where each element in the array has the identical format, all having the same field length, field type, and type parameter. Next, the FDODTA data object is built. It contains the data array for each field. Each field is repeated the number of times specified by the FDOEXT data object. Finally, the FDOOFF is built. It contains the offset values for each field. The offset value is the relative byte count from the start of the FDODTA data object to the first byte of the first element of each field. Offsets and extents are placed in the order as they occur in the associated FDODSC.

The SQLDTA data object contains the non-LOB input array data. If there is LOB data associated with a variable then the LOB values are externalized in EXTDTA data objects. EXTDTA objects flow after the SQLDTA command data. EXTDTA objects flow in database row order. The EXTDTAs for the first row flow in the order that the FD:OCA Generalized String headers occur in the associated SQLDTA. All LOB values for the first row (all LOB values defined in the first entry in each array) flow and then each subsequent row flows. The SQLDTA is allowed to be preceded by a TYPDEFNAM and/or TYPDEFQVR data object.

2. The server receives and processes each row in the array data in the EXCSQLSTT command and creates the appropriate reply. In this example, an SQLCARD reply data object is returned. The request executes passing the input arrays 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. An option on the SQL statement provides the database server when the requester wants the multiple-row request to succeed or fail as a unit, or if it wants the database server to proceed despite a partial (one or more rows) failure. The SQL clause to do this is ATOMIC or NOT ATOMIC where ATOMIC specifies that if the insert for any row fails, then all changes made to the database by any of the inserts, including changes made by successful inserts, are undone. This is the default. When NOT ATOMIC is specified, the inserts are processed independently. This means that if one or more errors occurs during the execution of an INSERT statement, then processing continues and any changes made during the execution of the statement are not backed out.

For each row that is processed by the database server, a condition is added to the SQLDIAGGRP. Each condition is needed to allow the requester to retrieve diagnostics information about each row processed. After the request is processed, the SQLCAGRP and SQLCAXGRP completion information is set to the following:

SQLCODE SQLCODE of last error.

SQLSTATE SQLSTATE of last error.

SQLERRD3 Actual number of rows processed.

SQLWARN Accumulation of flags set during any single insert.

Additionally, when NOT ATOMIC is in effect then status information is available for each failure or warning that occurred while processing the request. The status information for each row is available in the extended portion of the SQLCARD. 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.

3. The requester returns to the application with the indication of the success or failure of the request. The SQLCARD reply data object contains the SQLDIAGGRP that contains an array of conditions for each row processed by the database server.

### 4.4.8 Preparing an SQL Statement

Figure 4-34 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.

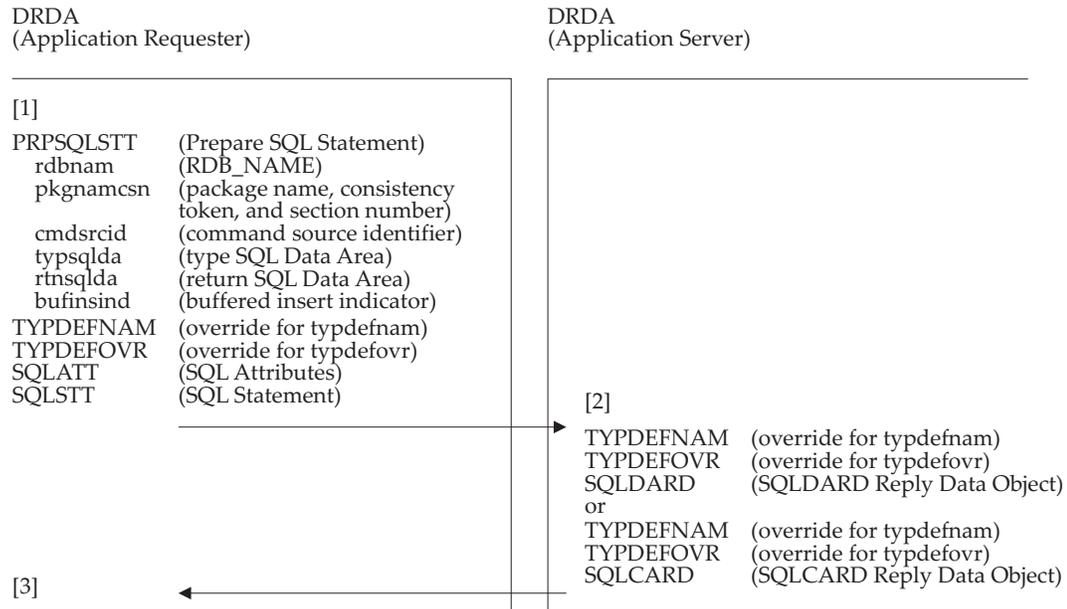


Figure 4-34 Preparing an SQL Statement

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 89)), 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. In addition, if the PRPSTTKP bind option was specified during the bind process, the statement can be executed across units of work.

**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, except if the PRPSTTKP option was specified during bind processing which keeps prepared statements across units of work.

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. The optional *cmdsrcid* parameter uniquely identifies the source of the command. 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 optional *bufinsind* parameter indicates to the server for an SQL INSERT statement what flavor of buffered insert, if any, should be used when the statement gets executed in an atomic multi-row input operation against a partitioned database.

The application requester places the SQL statement to be prepared into the SQLSTT command data object and optionally sends the command to the application server.

The *typsqlda* parameter identifies the level of describe information to be returned. A light, standard, or extended column of descriptive information can be requested. Refer to SQLDAGRP for details on the level of descriptive information that can be requested. If not specified, the standard describe information is returned.

2. The application server receives and processes the PRPSQLSTT command and SQLSTT and optionally the SQLATTR command data objects 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 unless the PRPSTTKP option was specified when the package was bound. The PRPSTTKP option allows prepared statements to be kept across units 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 an SQLDA with zero data variable definition entries 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 the statement is an SQL INSERT statement, the optional *bufinsind* parameter indicates to the server what flavor of buffered insert, if any, should be used when the statement gets executed in an atomic multi-row input operation against a partitioned database. This parameter has no effect and is ignored otherwise.

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 an SQLDA with zero data variable definition entries) indicating the error condition.

The level of SQLDARD generated is dependent on the level specified on the *typsqlda* parameter.

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 unless the PRPSTTKP option was specified when the package was bound. The PRPSTTKP option allows prepared statements to be kept across units 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 4-27](#) or creates and sends an OPNQRY command as described in step 1 of [Figure 4-20](#) (on page 148).

#### 4.4.9 Retrieving the Data Variable Definitions of an SQL Statement

Figure 4-35 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.

The EXCSQLSET command requires package name and consistency token parameters, but no section number parameter, as it is not bound into a package. Support for the SET CURRENT PACKAGE PATH statement is contingent on support of the EXCSQLSET command, as this value is propagated from a requester to a database server (possibly through intermediate servers) using the EXCSQLSET command.

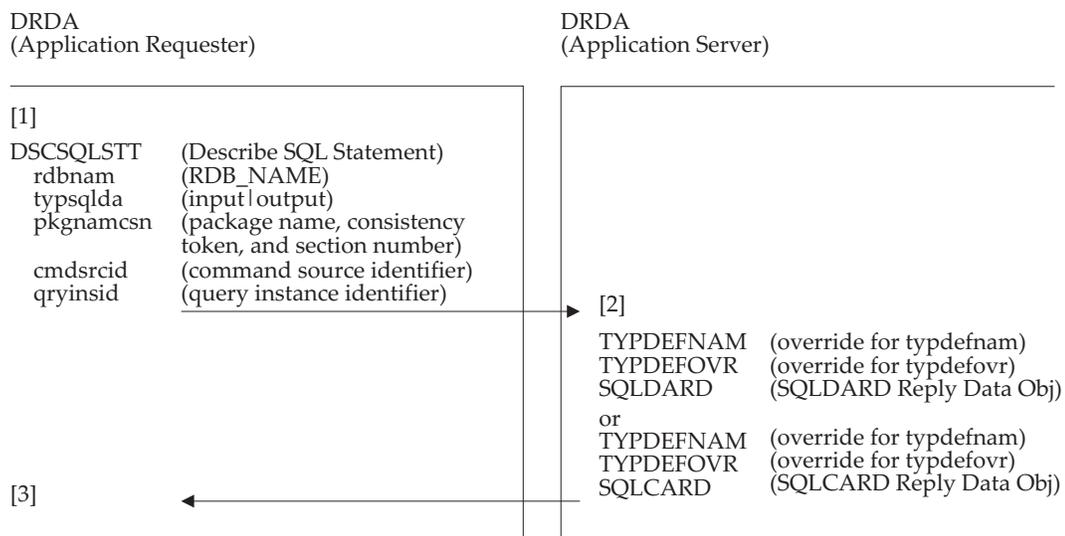


Figure 4-35 Describing a Bound SQL Statement

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 89)), 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. The optional *cmdsrcid* parameter uniquely identifies the source of the command. If this operation is performed on an SQL statement that has an open cursor associated with it, then the *qryinsid* value returned previously on the OPNQRYRM reply message must also be supplied on the DSCSQLSTT command. It then sends the command to the application server.

The *typsqlda* parameter identifies the level of describe information to be returned. A light, standard, or extended column of descriptive information can be requested. Refer to SQLDAGRP for details on the level of descriptive information that can be requested. If not specified, the standard describe information is returned.

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 an SQLDA containing zero data variable definition entries. In either case, the application server returns the SQLDARD reply data object to the application requester. In addition, in the case of an SQL error, the server may return an SQLCARD reply data object instead of an 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.

The level of SQLDARD generated is dependent on the level specified on the *typsqlda* parameter.

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 4-36 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.

The EXCSQLSET command requires package name and consistency token parameters, but no section number parameter, as it is not bound into a package. Support for the SET CURRENT PACKAGE PATH statement is contingent on support of the EXCSQLSET command, as this value is propagated from a requester to a database server (possibly through intermediate servers) using the EXCSQLSET command.

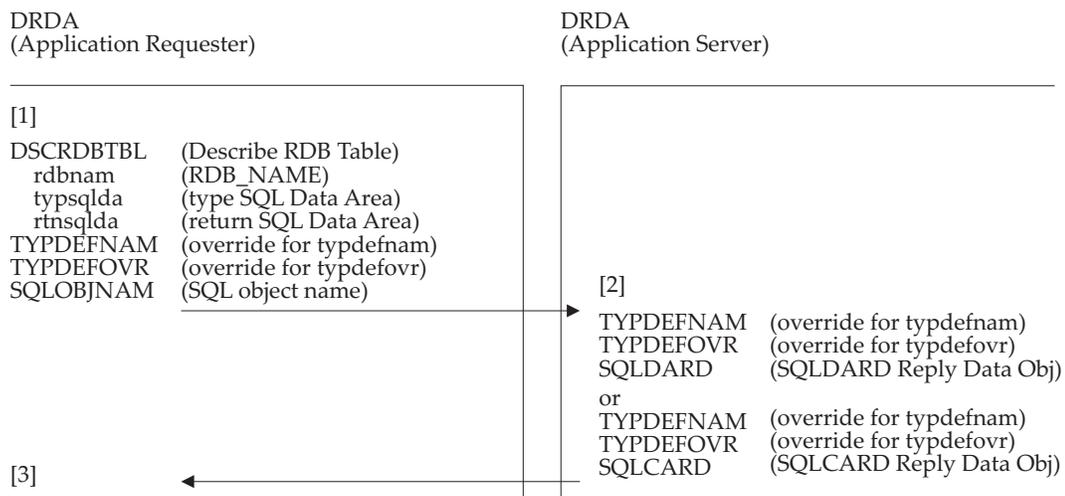


Figure 4-36 Describing a Table

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 89)), 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.

The *typsqlda* parameter identifies the level of describe information to be returned. A light, standard, or extended column of descriptive information can be requested. Refer to **SQLDAGRP** for details on the level of descriptive information that can be requested. If not specified, the standard describe information is returned.

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, exception conditions may be reported in an SQLCARD reply data object, or in an SQLDARD reply data object which contains an SQLDA with zero data variable definition entries.

**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.

The level of SQLDARD generated is dependent on the level specified on the *typsqlda* parameter.

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 4-37 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.

The EXCSQLSET command requires package name and consistency token parameters, but no section number parameter, as it is not bound into a package. Support for the SET CURRENT PACKAGE PATH statement is contingent on support of the EXCSQLSET command, as this value is propagated from a requester to a database server (possibly through intermediate servers) using the EXCSQLSET command.

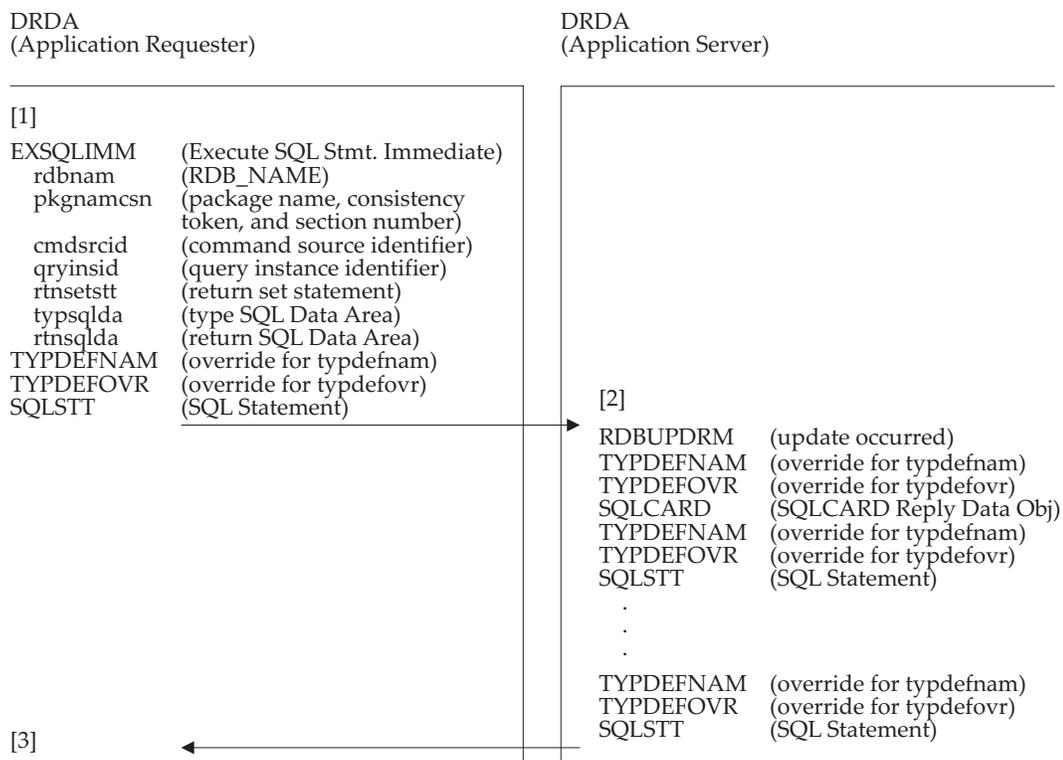


Figure 4-37 Immediate Execution of SQL Work

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 89)), the user can execute some SQL statements without binding them in a package (see Section 4.4.3 (on page 127)) or preparing them (see Section 4.4.8 (on page 195)). 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 *cmdsrcid* parameter uniquely identifies the source of the command. If the SQL statement being executed is a positioned delete/update, then the *qryinsid* parameter must also be specified in order to indicate the instance of the query in use, unless only a single query instance exists for the section. The optional *rdcbmtok* parameter informs the RDB whether or not it can process commit and rollback operations. The optional *rtissetstt* parameter specifies whether the server must return one or more SQL SET statements for any special registers whose settings have been changed on the current connection, if the execution of the command causes any special register setting to be updated. 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 any special register has been updated during execution of this command, as per the setting of the optional *rtissetstt* parameter, the server may return one or more SQLSTT reply data objects, each containing an SQL SET statement for a special register whose setting has been changed on the current connection. 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 4.4.15.1](#) and [Section 4.4.15.2](#) 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. The application requester also caches all SQL SET statements that may have been returned from the server in SQLSTT reply data objects so that they can be used later to restore the execution environment when the connection is reestablished to the database at either the original location or an alternate failover location in case of a communications failure.

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

#### 4.4.12 Returning SQL Diagnostics

The figure below indicates the DDM commands and replies that flow during the execution of an SQL statement when the RDB is accessed with diagnostics defined at Level 1. Diagnostics information is returned in the SQLCA reply data. The SQL GET DIAGNOSTIC statement does not flow. The diagnostic group is optional and must be requested using the DIAGLVL instance variable when accessing the remote RDB. Servers do not have to provide the diagnostics group at the completion of an SQL statement unless the command is performing a multi-row operation. Multi-row operations can return multiple error and warning conditions that cannot be returned unless the new diagnostic object is provided.

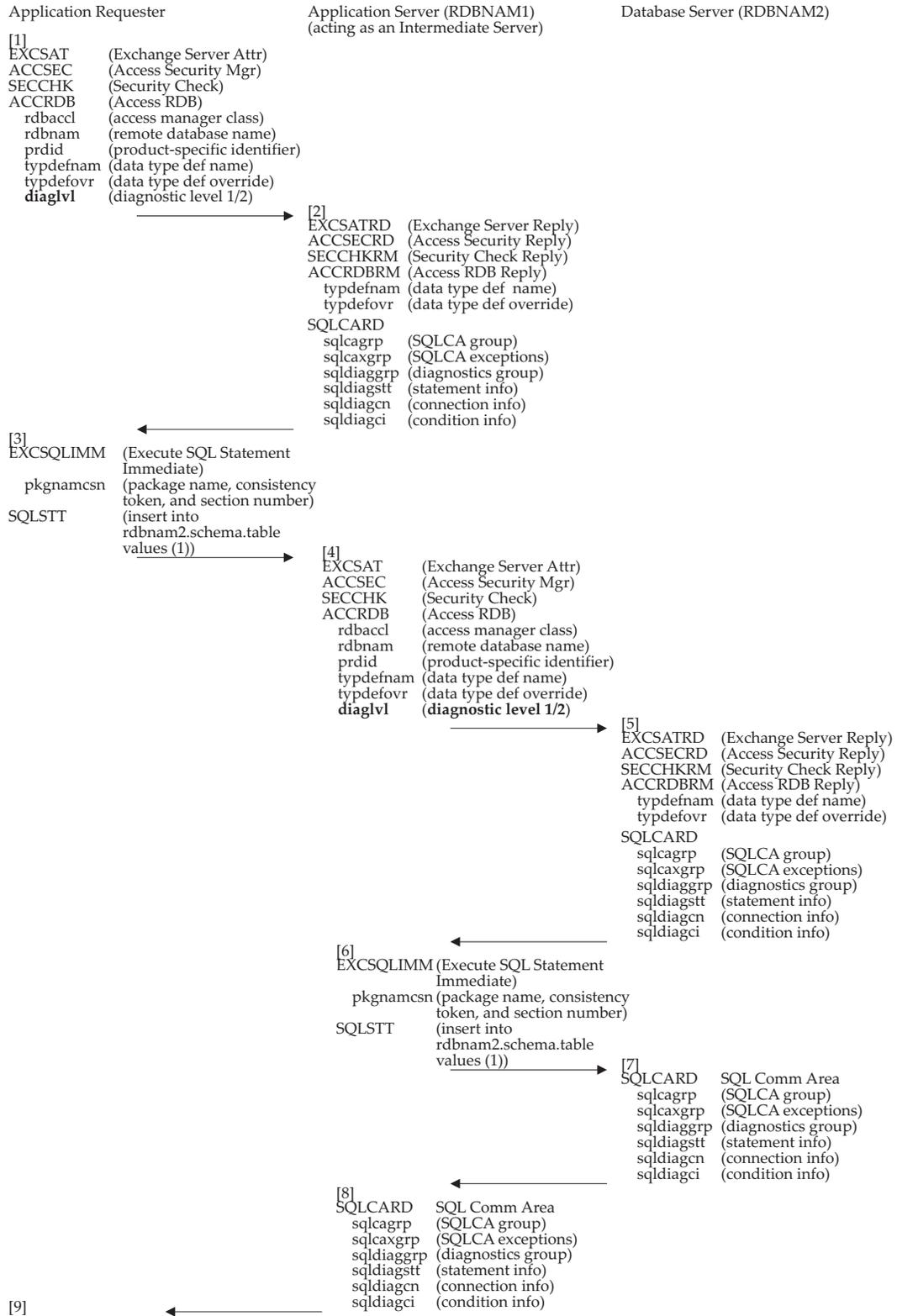


Figure 4-38 Returning SQL Diagnostics

1. An application issues the SQL CONNECT statement to access RDB RDBNAM1. The application requester resets the local diagnostic area and then accesses the remote RDB requesting diagnostic Level 1 or 2.
2. The server authenticates the user, creates a process to process requests, and builds a diagnostic area. Optionally, it can build an SQLCARD to return diagnostics for the SQL CONNECT.
3. If the reply does not contain the optional SQLCARD, the requester generates the connection information for the SQL CONNECT statement based on information provided on the EXCSAT, SECCHK, ACCSEC, and ACCRDB commands and replies. The application can then issue SQL GET DIAGNOSTICS to get information about the statement connection and any conditions generated processing the SQL CONNECT statement. The application then executes an SQL immediate statement to insert data into RDB RDBNAM2. The application requester builds and sends the EXCSQLIMM with the SQL statement text.
4. The server processes the EXCSQLIMM command. It parses the SQL statement text and determines the statement for a remote server. The application server acting as an intermediate server resets the local diagnostic area and then accesses the remote RDB requesting diagnostic Level 1 or 2. The connection information for the EXCSQLIMM is added to the local diagnostic area from the information generated during the execution of the EXCSAT, ACCSEC, SECCHK, and ACCRDB commands.
5. The server authenticates the user, creates a process to process requests, and builds a diagnostic area. Optionally, it builds an SQLCA to return statement information, connection information, and any conditions generated during the processing of the EXCSAT, SECCHK, and ACCRDB commands.
6. If the SQLCA is not returned, the server generates the connection information based on information returned on the EXCSAT, ACCSEC, and ACCRDB replies. It adds the connection information to the diagnostic area for the EXCSQLIMM command. Conditions may be added to the local diagnostics if warnings occur during the processing of the EXCSAT, ACCSEC, SECCHK, and ACCRDB replies. The intermediate server then sends the EXCSQLIMM with the SQL statement text.
7. The server receives and processes the EXCSQLIMM command. During the execution of the statement, diagnostics are collected in the diagnostic area during the execution of the statement. At completion of the statement, the server builds the SQLCARD with the diagnostics group. The diagnostic group contains statement information for the SQL INSERT command, a null connection array, and the condition array with any warning or error condition entries generated during the execution of the statement. The server returns the SQLCARD to the intermediate server.
8. The server receives and processes the SQLCARD reply. A connection entry is appended to the connection array in the diagnostic area. Condition entries are appended to any existing warning entries generated during the processing of the statement. If DIAGLVL1 is specified, one of the SQLDCMSG fields should contain the message text for the condition. If DIAGLVL2 is specified, the SQLDCMSG fields should be NULL. The SQLCARD is generated from the diagnostic area which contains statement information for the SQL INSERT command, one connection entry, and the condition array with any warning or error condition entries generated during the processing of the statement. The intermediate server returns the SQLCARD to the application requester.
9. The application requester receives the SQLCARD reply. The SQLCARD is parsed. The statement information for the SQL INSERT statement is added to the diagnostic area. The connection information is appended to the connection array. In this case two connection

entries exist in the diagnostic area. Conditions are appended to any existing warning entry conditions in the condition array generated during the local processing of the statement. If the application issues the GET DIAGNOSTICS statement, the local diagnostic area contains all the diagnostics required to process the statement.

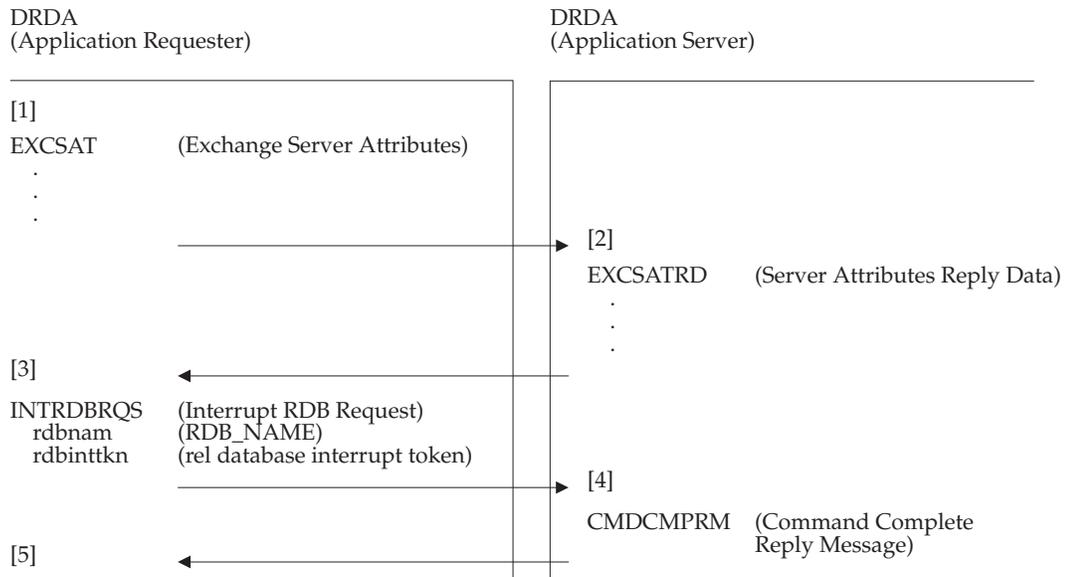
#### 4.4.13 Controlling the Amount of Describe Information Returned

The early SQL Descriptor Area defines an area that provides descriptive information for each input variable and output variable related to an SQL statement. An SQL DESCRIBE, an SQL EXECUTE, an SQL CALL, an SQL OPEN, or an SQL PREPARE statement may require different kinds of descriptive information to be provided to an application depending on the type of application. For example, some dynamic SQL applications (that is, JDBC) may require more descriptive information than static SQL applications. To allow a requester to control the amount of describe information to be generated and returned by the server, a light descriptor, a standard descriptor, or an extended descriptor area can be requested. The DDM TYPSQLDA instance variable on the DDM DSCRDBTBL (DESCRIBE TABLE), DDM DSCSQLSTT (DESCRIBE STATEMENT), DDM EXCSQLSTT (CALL or EXECUTE), DDM OPNQRY (OPEN), or DDM PRPSQLSTT (PREPARE) commands controls the type of descriptor area to be contained in the SQLDARD. The RSLSETFLG controls the type of descriptor area to be contained in the SQLCINRD.

If the requester requests a light descriptor, the SQL Descriptor Header group, the SQL Descriptor Optional group, the SQL Descriptor UDT group, and the SQL Descriptor Extended group are returned as null groups. For a standard descriptor, the SQL Descriptor Header fields are returned as null, the SQL Descriptor Optional fields are returned and the group must not be null, the SQL Descriptor UDT group fields are returned if the variable or column described is a user-defined data type, and the SQL Descriptor Extended group must be null. For an extended descriptor, all the SQL Descriptor groups are returned. The SQL Descriptor Header fields are returned, the SQL Descriptor Optional fields are returned, the SQL Descriptor UDT fields are returned if the variable or column described is a user-defined data type, and the SQL Descriptor Extended fields are returned. Refer to the identified group descriptor for the type of descriptive information returned by each group.

#### 4.4.14 Interrupting a Running DRDA Request

Figure 4-39 indicates the DDM commands and replies that flow when a user or process requests the interrupt of a running DRDA request.



**Figure 4-39** Requests to Interrupt DRDA Requests

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 command. For a detailed description of the parameters, see the DDM Reference.

- 1.&2. After the application request establishes a separate connection using the same communications manager as well as the same communication settings as the connection to be interrupted, the application requester and the application server exchange attributes. The required managers on EXCSAT should be at DDM Level 5.
3. The application requester sends the INTRDBRQS command to the application server to request an interrupt of the DRDA request running on the other connection. See Figure 4-40 for the flow of an Interrupted DRDA request. The application requester received the value for the *rdbinttkn* parameter in the ACCRDBRM reply message at access relational database time for the connection running the DRDA request to be interrupted. This value must be available at the application requester.

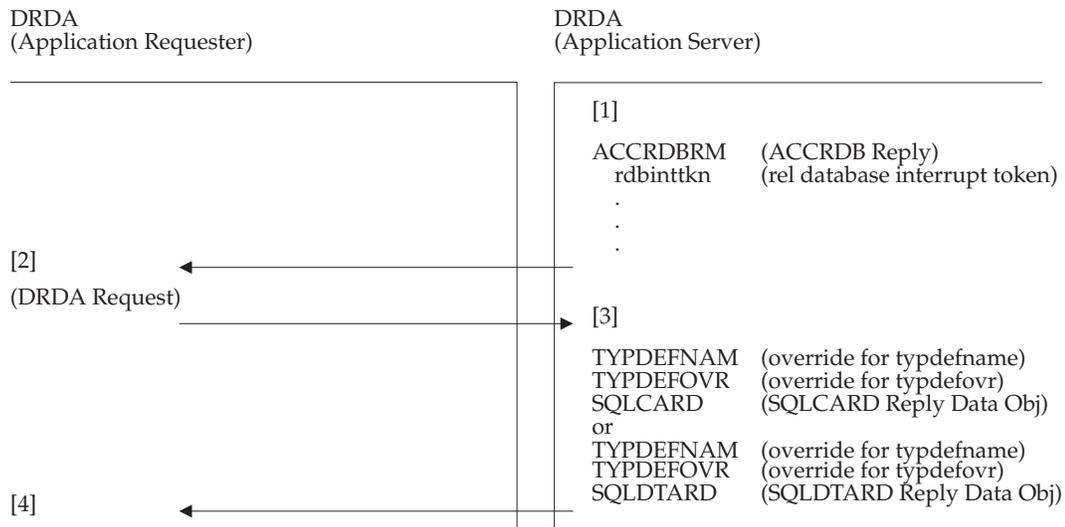
**Notes:**

1. The INTRDBRQS command is not valid after an Access Relational Database command (ACCRDB) on the same connection.
2. The ACCRDB command is not valid after an INTRDBRQS command on the same network connection.
3. Multiple INTRDBRQS commands, however, may be sent on the same network connection.
4. The application server receives and processes the INTRDBRQS command and creates a reply message. After validity checking, the application server sends a CMDCMPRM reply message back to the application requester to indicate a successful completion. The

application server will send this message even if the actual interrupt has not taken place. (The database manager may not be in an interruptible state or the process may already be finished.) See [Figure 4-40](#) for a flow of an interrupted DRDA request.

- When the application requester receives the reply message, the connection is de-allocated. [Figure 4-40](#) indicates the DDM commands and replies that flow when a DRDA request is interrupted.

[Figure 4-40](#) indicates the DDM commands and replies that flow when a DRDA request is interrupted.



**Figure 4-40** Interrupted DRDA Request

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.

- After the application requester and the application server establish a connection to execute DRDA requests, the application server returns a token in the *rdbinttkn* parameter of the **ACCRDBRM** reply message. This token is for interrupting a DRDA in this connection. If an application server does not support the interrupt function, it will not return this parameter.
- When the connection is established, SQL requests can begin flowing.
- When an **INTRDBRQS** command executes against a DRDA request in this connection (see [Figure 4-39](#) (on page 209)), it aborts execution of the DRDA request. This application server returns an **SQLSTATE** of 57014 in the **SQLCA** for the interrupted DRDA request. The database manager returns to the state it was in prior to execution of the DRDA request. This does not imply the database manager is committed or rolled back.

**Notes:**

1. The interrupt does not affect the execution of a commit or rollback function.
  2. If there is no DRDA request executing, the interrupt is ignored.
  3. If the interrupt occurs during processing of an OPNQRY or CNTQRY, an ENDQRYRM and an SQLCARD are returned.
  4. If the interrupt occurs during processing of a BNDSQLSTT statement, an SQLERRM and SQLCARD are returned.
  5. It is possible that the command being interrupted will have almost completed before it receives the INTRDBRQS command. DRDA can allow the original command to complete and to ignore the interrupt. DRDA does not define almost condition.
  6. It is possible that the command being interrupted cannot be interrupted, in which case the interrupt is ignored.
  7. The application server returns the SQLCA for the interrupted request using the normal carrier for that request.
4. The application requester receives the response from the application server and informs the application of the condition.

#### 4.4.15 Commitment of Work in DRDA

An application requester that is not using the XAMGR normally initiates commit or rollback processing by either calling the sync point manager, or by executing a COMMIT or ROLLBACK SQL statement. An application requester that is using the XAMGR must perform two-phase commit against each resource updated within the transaction.

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. See Rule CR9 in [Section 7.5](#) for exceptions on 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 application requester is using the services of the XAMGR, and it terminates normally, but does not explicitly commit (two-phase) or rollback, then the application server must follow the presumed abort protocol and initiate rollback. 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. If the application requester is using the services of the XAMGR, and it terminates abnormally, the application server must follow the presumed abort protocol and initiate rollback.

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. The application server must inform the application requester the result of all transactional processing requests, whether successful or not.

Deadlocks or abnormal ending conditions at the application server can also cause rollback processing at the application server. If the application requester is using the services of the sync point manager at Level 7, then the application can share recoverable resources at the application server, so as to prevent deadlocks. The XA manager will share recoverable resources automatically as specified by *DTP: The XA+ Specification* dealing with tightly and loosely-coupled transactions. The XAFLAG(TM\_LCS) flag must also be sent to enable loosely-coupled transactions; see the DDM Reference, XAMGROV for more details.

Within DRDA environments, all forms of commit and rollback requests are equivalent.

In DRDA, the application requester that is not using the services of the XA manager 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. When using the services of the XAMGR, the application is responsible for the management of all XAMGR protected connections, for the management of commitment and rollback processing, for the management of update privileges, and also for ensuring complete data integrity of all resources involved within that transaction. The application is also responsible for proper termination of these connection.

## 4.4.15.1 Commitment of Work in a Remote Unit of Work

Figure 4-41 indicates the DDM commands and replies that flow to commit the unit of work on DRDA Level 3 connections. Figure 4-42 indicates the flows when commit is included in a stored procedure on a DRDA Level 1 application server.

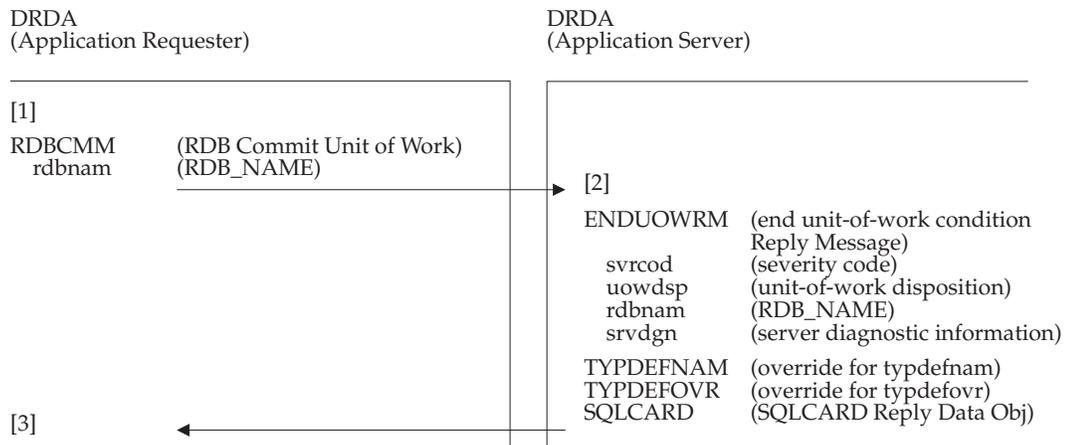


Figure 4-41 Commit a Remote Unit of Work

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 202), 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 195), 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).

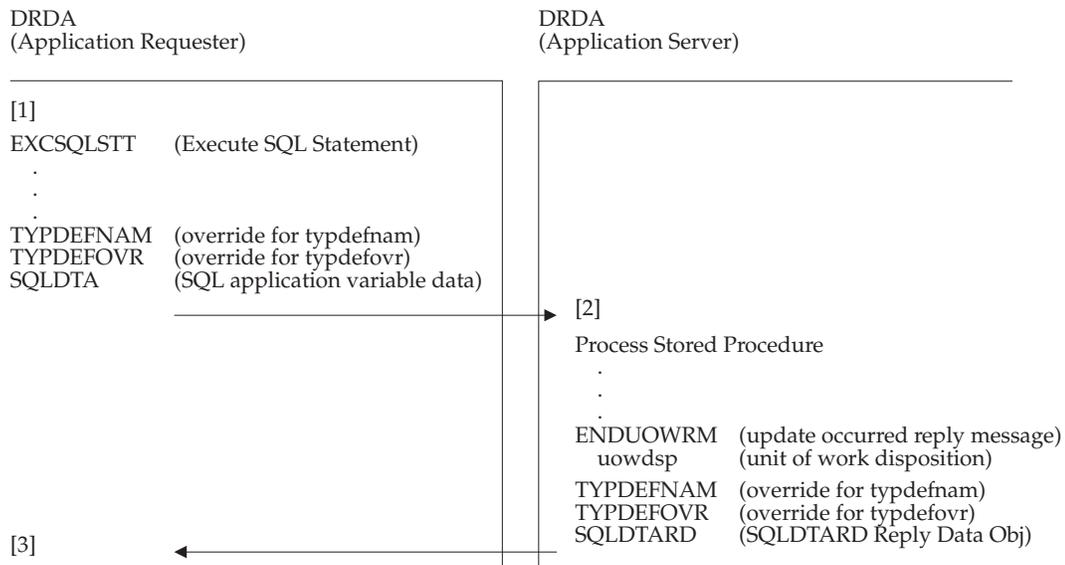
Otherwise, the SQLCARD reply data object indicates a single error. The application server returns the ENDUOWRM reply message and the SQLCARD to the application requester and rolls back the unit of work.

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 4-42 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.



**Figure 4-42** 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](#) for details.

The application requester also performs cursor management operations dependent on the value of the `uowdsp` parameter.

#### 4.4.15.2 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.

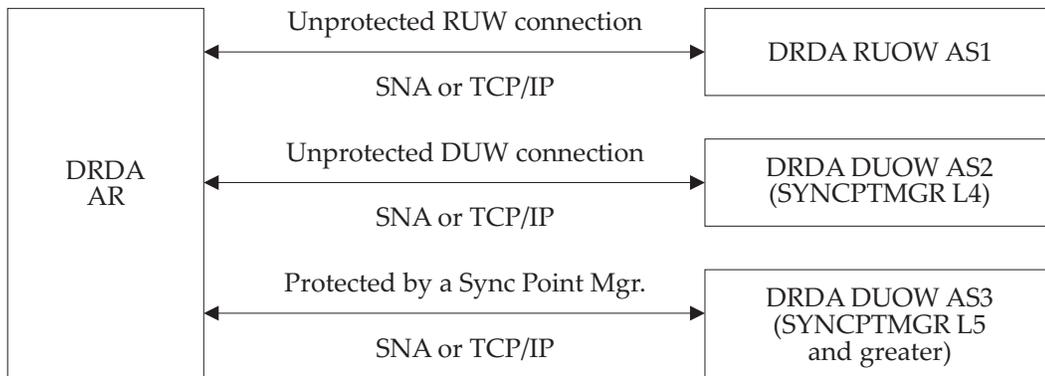
The Multi-Relational Database Update environment is where the services of a sync point manager are required to perform resource recovery for the unit of work. Because of this, all application servers that are on network connections protected by two-phase commit protocols have update privileges. All application servers that are not on network connections protected by two-phase commit protocols are restricted to read-only.

Application servers that share recoverable resources between a set of protected connections, must prevent deadlocks from occurring between these connections. The degree of sharing depends on whether the application requires partial or complete sharing. For partial sharing the RDB will examine the XID of all the SYNCPTMGR protected connections. The set of SYNCPTMGR protected connections whose XIDs match exactly, will share recoverable resources between themselves so as to prevent any deadlocks from occurring. If the application has requested complete sharing, then the RDB will only examine the XIDs *gtrid* value (see XID in the DDM Reference for format details). The set of SYNCPTMGR protected connections whose *gtrid* part of the XID match will share resources between themselves so as to prevent any deadlocks from occurring.

Figure 4-43 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 4-43 (on page 218), 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 4-43** 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 4-44 is an example of this flow for EXCSQLIMM.

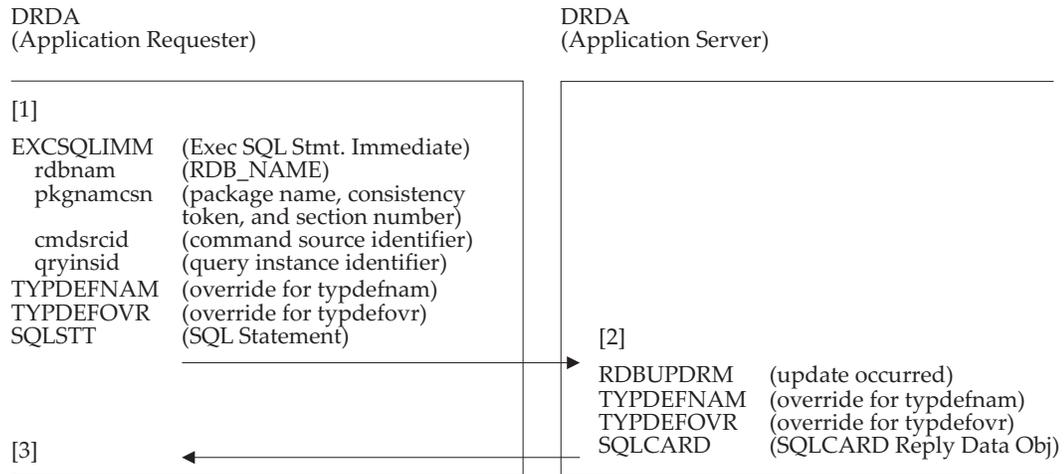


Figure 4-44 DRDA RDBUPDRM Example Flow

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.

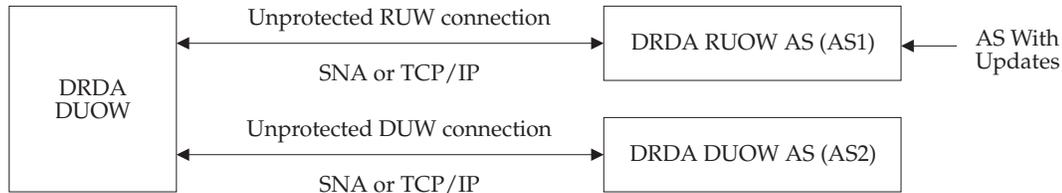
If a commit or rollback request is application-directed to a read-only application server operating in a Distributed Unit of Work environment, a DDM reply message CMMRQSRM, with the *cmmtyp* parameter indicating a commit or rollback, is returned to the application requester. If the local environment 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 4-45 (on page 221)) on an unprotected network connection.



**Figure 4-45** Single RDB Update at a DRDA Remote Unit of Work Application Server

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 4-37 (on page 202). The EXCSQLSTT command flow is described in Figure 4-27 (on page 172).
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 4-41 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 4-37](#) (on page 202). The EXCSQLSTT command flow is described in [Figure 4-27](#) (on page 172).
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.

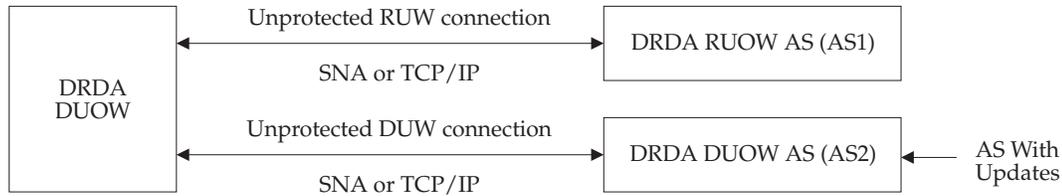
Regardless of the outcome from the read-only application servers, the result returned to the application must reflect the status of the work at the update application server.

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 4-46 (on page 225)). 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 4-46** 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 4-37 (on page 202). The EXCSQLSTT command flow is described in Figure 4-27 (on page 172).
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 4-37](#) (on page 202). The EXCSQLSTT command flow is described in [Figure 4-27](#) (on page 172). 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 4-47 (on page 229)). 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.

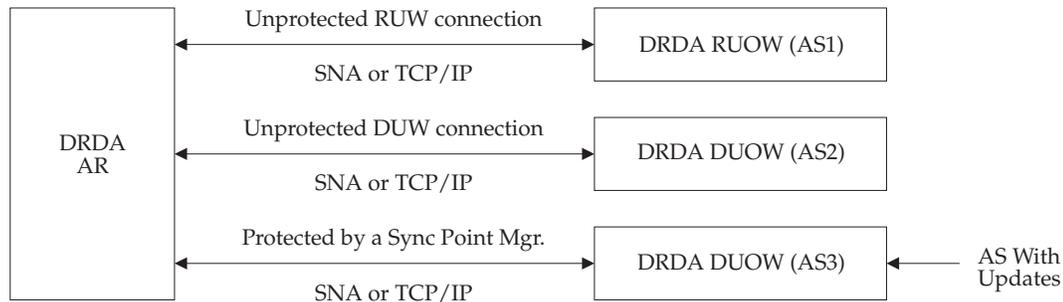


Figure 4-47 Multi-Relational Database Update

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 4-37 (on page 202). The EXCSQLSTT command flow is described in Figure 4-27 (on page 172).
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 4-37](#) (on page 202). The EXCSQLSTT command flow is described in [Figure 4-27](#) (on page 172).
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.

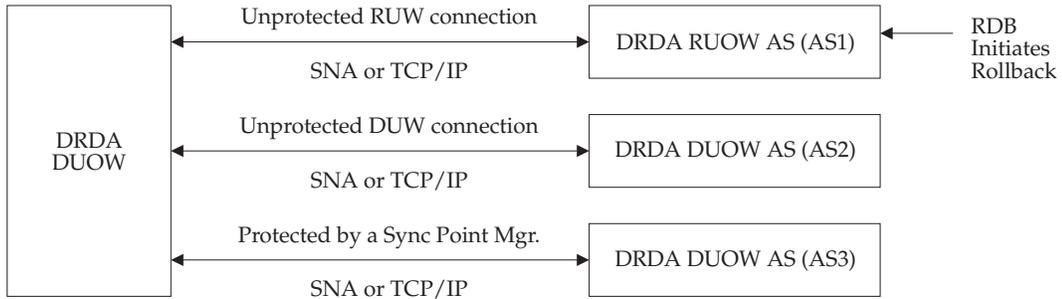
### Post-Commit Processing

After the commit processing completes as defined in [Commit and Rollback Scenarios](#) (on page 219), 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 217).

**RDB-Initiated Rollback**

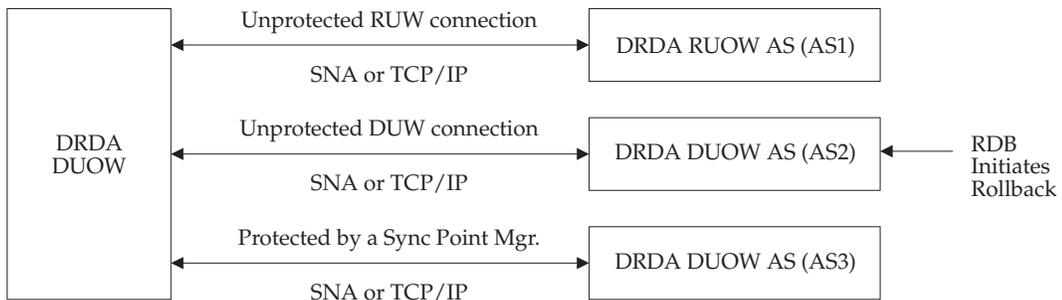
In the following scenarios, a relational database has initiated rollback. A relational database initiated rollback is due to an uncontrollable event on the relational database that requires it to roll back immediately. The following scenarios describe the steps that occur, depending on which application server initiates the rollback. [Figure 4-48](#) and [Figure 4-49](#) show unprotected network connections between the application requester and application servers.



**Figure 4-48** RDB at AS1 Initiates Rollback

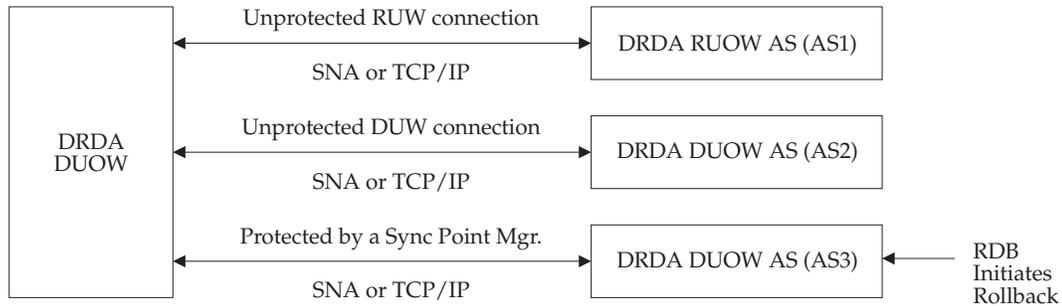
The network connection between the application requester and AS1 is unprotected, so there is not a sync point manager involved at AS1.

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 219). The application requester returns to the application the SQLCA received from AS1.



**Figure 4-49** 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 4-50** 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 SYNCRRD 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 SYNCRRD 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 219). 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'.

#### 4.4.15.3 Global and Local Transactions

All transactions on an XAMGR protected connection must be identified by a Transaction Identifier. The identifier must always be registered with the DBMS before any work is performed, and ended with the DBMS after the work is completed. The identifier represents the unit of work that was performed on the connection between the start and end blocks. The application uses this identifier to coordinate and recover transactions. The application must associate each distributed unit of work or transaction with an XID, modeled after XA's XID. The XID has two portions, the GTRID and the BQUAL.

The GTRID is the Global Transaction identifier, and represents the distributed unit of work or transaction in its entirety. Each Global Transaction may have one or more DBMSs within the transaction. Work occurring anywhere in the system must be committed atomically. The application must coordinate the DBMS's recoverable unit of work that is part of a Global Transaction.

The BQUAL is the Branch Qualifier, and represents one or more transaction branches in support of a Global Transaction for which the application will engage in a separate but coordinated transaction. Each DBMS's internal unit of work in support of a Global Transaction is part of exactly one branch. A Global Transaction may have more than one branch; for example, a Global Transaction may span one or more DBMSs. The application may associate a different branch to each DBMS within the Global Transaction. All branches are related in that they must be completed atomically; however, the application coordinates each branch separately.

In addition to the XID, a flag representing lock sharing will also be transmitted to the RDB. The flag requires the RDB to optimize shared resources and locks to prevent any deadlocks from occurring between connections involved in the same Global Transaction but with different branches. If the flag is not sent, the RDB is required to treat each XID as a unique transaction. If the RDB cannot support sharing locks and resources, than it should ignore the flag and revert to its default behavior regarding locks (see XAMGROVR for more details).

An application may want to perform an unprotected transaction on an XAMGR protected connection. Such transactions are known as Local Transactions, and require no ending, two-phase coordination, or recovery. Local Transactions behave exactly like a transaction over a non-protected connection. The first SQL statement initiates the local transaction at the DRDA application server. At any given time, the XA protected connection can either be IDLE, in a Global Transaction, or in a Local Transaction. The application then issues an RDBCMM or RDBRLLBCK to end the Local Transaction. At which point, the connection can be reused to start another Local Transaction or Global Transaction. Attempts to use SYNCCTL to commit or rollback the transaction are a protocol violation. If the application terminates normally but does explicitly commit or roll back the Local Transaction, the XAMGR will drive an implicit rollback. If the application terminates abnormally, than the DBMS will drive an implicit rollback.

The following diagrams show high-level flows illustrating Global and Local Transactions. [Table 4-4](#) shows the steps involved in a Global Transaction. The first step is to open the connection requesting the services of the XA manager. The application server responds whether or not it can support the XA manager. Once the connection has been established—that is, security check and RDB access—the Global Transaction can now begin (Step 2). To start the Global Transaction, a SYNCCTL(New UOW) request is sent to the application server with the Transaction Identifier (XID). The application server passes this information to the DBMS to register the transaction, and responds to the New Unit of Work request.

The application requester issues work on the connection (Step 3). This work will be associated with the XID. The application requester than informs the application server that the Unit of Work is completed (Step 4). This is done by sending a SYNCCTL(End Association) request. The application server will pass this information to the DBMS, which will end the transaction. At

this point the application requester may start another Global Transaction (repeat Steps 2 to 4), start a Local Transaction (see next diagram), or commit/rollback the transaction. The commit process requires the use of the two-phase protocol; the application requester would send the SYNCCTL(Prepare to Commit) request to begin the first phase of the protocol (Step 5). Once the DBMS has written the log record, the application server responds to the SYNCCTL(Prepare to Commit) request. At this point the application requester can start the second phase (Step 6). This is done by sending the SYNCCTL(Commit) request to the application server.

**Table 4-4** Example of Global Transaction

DRDA (Application Requester)	DRDA (Application Server)
1. Open Conversation	Accept connection
EXCSAT (Exchange Server Attributes, XAMGR 7)	→ ← EXCSATRD (XAMGR7)
ACCSEC (Access Security) SECCHK (Security Check) ACCRDB (Access RDB)	→ ACCSECRD (ACCSEC Reply) SECCHKRM (SECCHK Reply) ← ACCRDBRM (ACCRDB Reply)
2. Start Global Transaction	
SYNCCTL (New Unit of Work: SYNCTYPE(X'09') XID XAFLAGS TIMEOUT)	→ ← SYNCCRD (SYNCCTL Reply: XARETVAL - XA return code)
3. Execute Work (SQL)	
EXCSQLIMM (Execute Immediate)	→ ← RDBUPDRM (RDB Update) SQLCARD (SQLCA)
4. End Global Transaction	
SYNCCTL (End Association: SYNCTYPE(X'08') XID XAFLAGS RLSCONV)	→ ← SYNCCRD (SYNCCTL Reply: XARETVAL)
5. First Phase of Two-phase Commit Protocol	
SYNCCTL(Prepare to Commit: SYNCCTYPE(X'01') XID XAFLAGS	

DRDA (Application Requester)	DRDA (Application Server)
RLSCONV)	→ ← SYNCCRD (SYNCCTL Reply: XARETVAL)
6. Second Phase of Two-phase Commit Protocol	
SYNCCTL (Commit: SYNCCTYPE(X'03') XID XAFLAGS RLSCONV)	→ ← SYNCCRD (SYNCCTL Reply: XARETVAL)

Table 4-5 shows an example of a Local Transaction. The difference between a Local and Global transaction is:

1. The XID that represents a Local Transaction always has a FormatID equal to -1 (see XID in the DDM Reference for more details).
2. No coordination is required during a commit/rollback process for Local Transactions.
3. No recovery is required for Local Transactions.

The application requester establishes a connection with XA manager support (Step 1), just like a Global Transaction. At this point (Step 2), the application requester begins a Local Transaction by sending the first SQL statement. Because this is an XAMGR protected connection, the application server puts itself into Local Transaction state and should not expect an End Association or two-phase coordination. The application requester then continues to issue work on the connection (Step 3). When the application requester is ready to commit the unit of work, it sends the services of the RDB manager.

After the commit, the application requester can begin any type of transaction; that is, a Local (repeat the above steps) or Global (use the above diagram).

**Table 4-5** Example of Local Transaction

DRDA (Application Requester)	DRDA (Application Server)
1. Open Conversation	Accept connection
EXCSAT (Exchange Server Attributes, XAMGR 7)	→ ← EXCSATRD (XAMGR 7)
ACCSEC (Access Security) SECCHK (Security Check) ACCRDB (Access RDB)	→ ← ACCSECRD (ACCSEC Reply) SECCHKRM (SECCHK Reply) ACCRDBRM (ACCRDB Reply)
2. Start Local Transaction (1st SQL)	
EXCSQLIMM (Execute Immediate)	→ ← RDBUPDRM (RDB Update) SQLCARD (SQLCA)

<b>DRDA (Application Requester)</b>		<b>DRDA (Application Server)</b>
3. Execute Work (SQL)		
EXCSQLIMM (Execute Immediate)	→	
	←	RDBUPDRM (RDB Update) SQLCARD (SQLCA)
4. Commit Local Transaction		
RDBCMM (RDB Commit)	→	
	←	ENDUOWRM

### 4.4.16 Connection Reuse

#### 4.4.16.1 Connection Pooling

Allows an application requester to reuse a network connection for a different application once an application disconnects from the connection (the application terminates or releases the connection). Thus, this level of connection reuse requires the connection flow to authenticate the new user and establish the connection environment as if a new connection is being established. Refer to the Connection Allocation (CA) and Control Usage (CU) rules for the rules that must be obeyed regarding connection reuse when the requester is performing connection pooling.

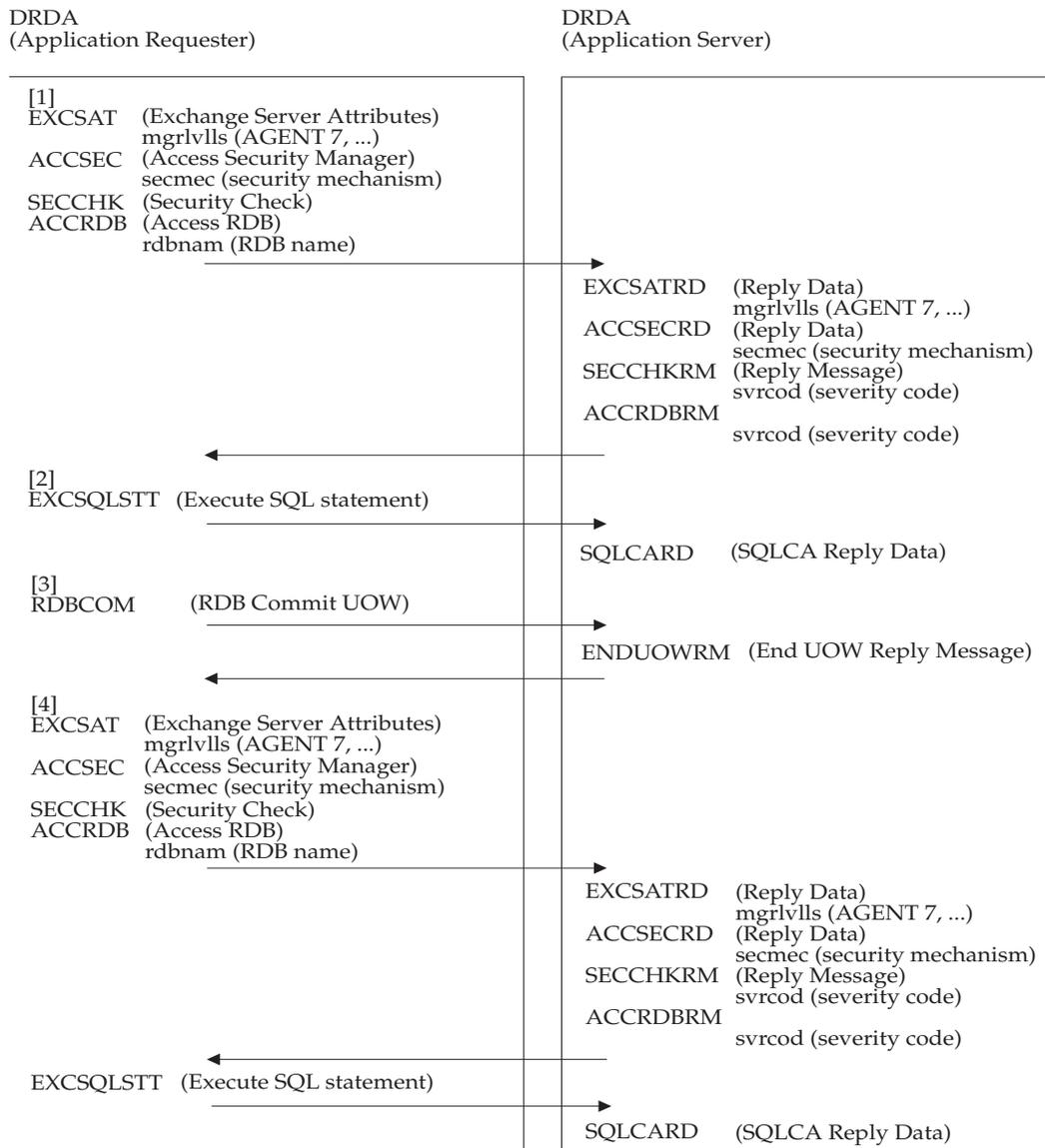


Figure 4-51 Reuse Connection using Connection Pooling

The following is a brief description of some of the parameters for the DDM commands for an unprotected connection using 1-phase commit. This example illustrates the flow when all DDM commands used to initialize a connection are chained together to minimize the network costs. The DDM Reference provides a detailed description of the parameters and chaining requirements.

1. The application requests a connection. The application requester establishes a network connection (described in [Chapter 12](#) and [Chapter 13](#) which describe Network Protocols) with the application server. The application requester issues the commands to access the RDB. The EXCSAT requests AGENT manager Level 7 which is required to perform connection pooling. The application server returns the replies for the connection supporting AGENT manager Level 7 indicating connection pooling is supported.
2. The application requester requests the execution of an SQL statement for the connected application. The application server replies with the SQL communications area.
3. The application requester commits the unit of work. The application server returns the end unit of work condition and the SQL communications area.
4. The application then terminates and the application requester keeps the existing connection to be reused by another application. When another application connects with the same network requirements as with the existing connection, the application requester again issues the commands in order to establish the connection with the RDB for the new application. When the connection commands are re-issued on the existing connection, the application server must close and destroy all RDB resources associated with the previous application. All held cursors are closed. Any RDB temporary tables are destroyed. The application server execution environment is initialized as if for a new connection.
5. The application requester requests the execution of an SQL statement for the connected application. The application server replies with the SQL communications area.

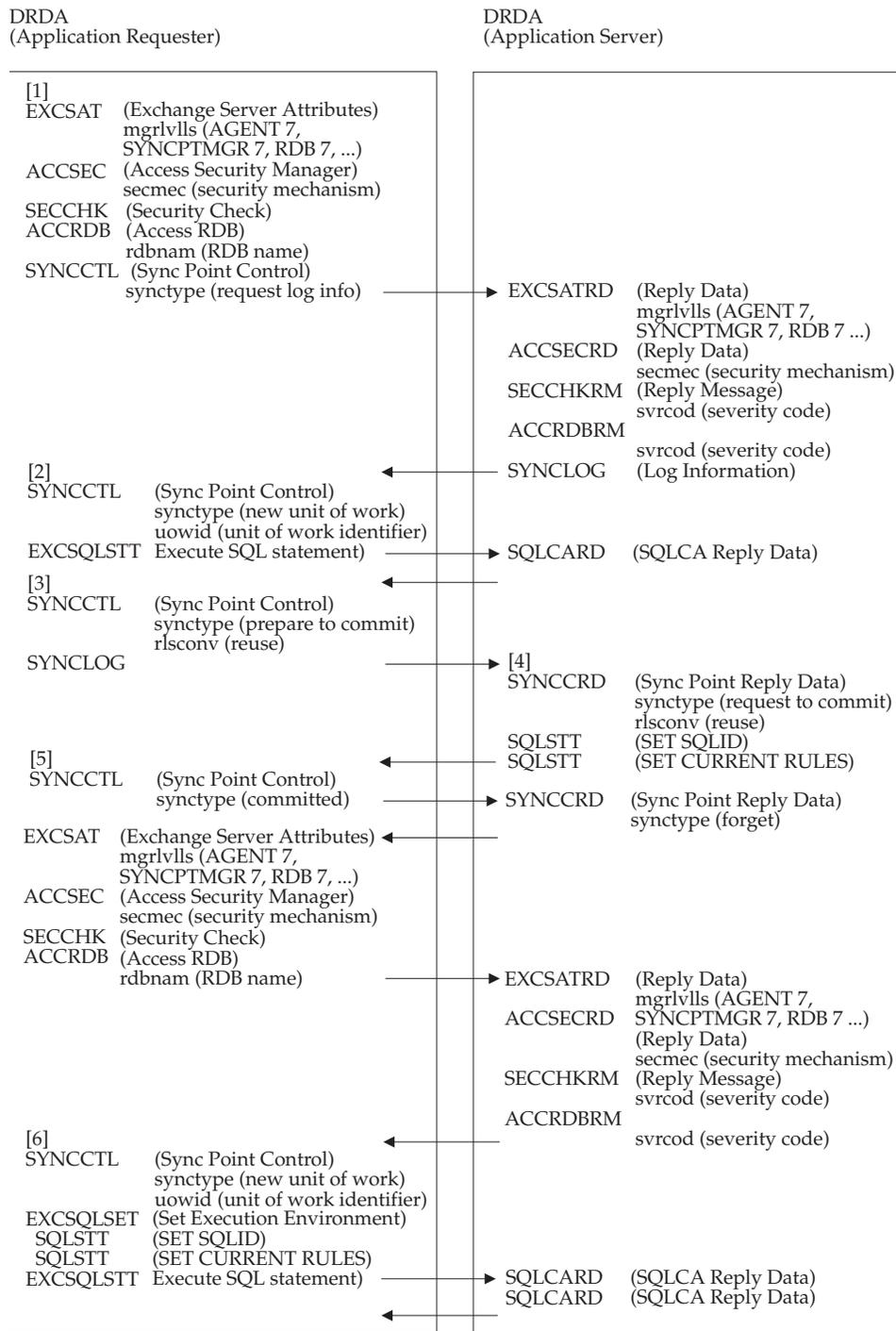
#### 4.4.16.2 Transaction Pooling

Transaction pooling allows an application requester to share a network connection with other applications after completion of a transaction. Transactions are delimited by a commit, rollback, or the dissociation of a transaction from the connection at SYNCCTL(End). An application may establish environment variables required by a transaction on the server that only the RDB knows about. As a result, if the application requester were to temporarily relinquish control of the connection to another application and then reuse the connection for the original application to resume its processing, environment variables and other RDB resources may have changed causing undeterministic results. The release connection parameter set to reuse on the SYNCCTL, RDBCMM, or RDBRLLBCK command is required to request the server to report environment information to the requester and to indicate whether the connection can be reused to execute a transaction for another application. The release connection parameter indicating reuse can be specified on any remote unit of work or distributed unit of work connection. If the server determines the connection can be reused, the release connection parameter is returned set to reuse; otherwise, the release connection parameter is returned set to false. Refer to the Connection Allocation (CA Rules) in [Section 7.2](#) and Connection Usage (CU Rules) in [Section 7.6](#) for the rules that must be obeyed regarding connection reuse when the requester is performing transaction pooling.

The table below shows the DDM requests that represent the transaction boundary for all connection types.

<b>Connection Type</b>	<b>Transactional Boundary (DDM Request)</b>
RUOW - Remote Unit of Work	RDBCMM - Commit RDBRLLBCK - Rollback
SYNCPTMGR protected connections	SYNCCTL(Commit) SYNCCTL(Rollback)
XAMGR protected connections in a Global Transaction	SYNCCTL(End) with flag TMSUCCESS, TMFAIL, or TMSUSPEND (if application server can support dissociation of transaction)
XAMGR protected connections in a Local Transaction	RDBCMM - Commit RDBRLLBCK - Rollback

**Pooling DUOW Network Connections**



**Figure 4-52** Reuse DUOW Connection using Transaction Pooling

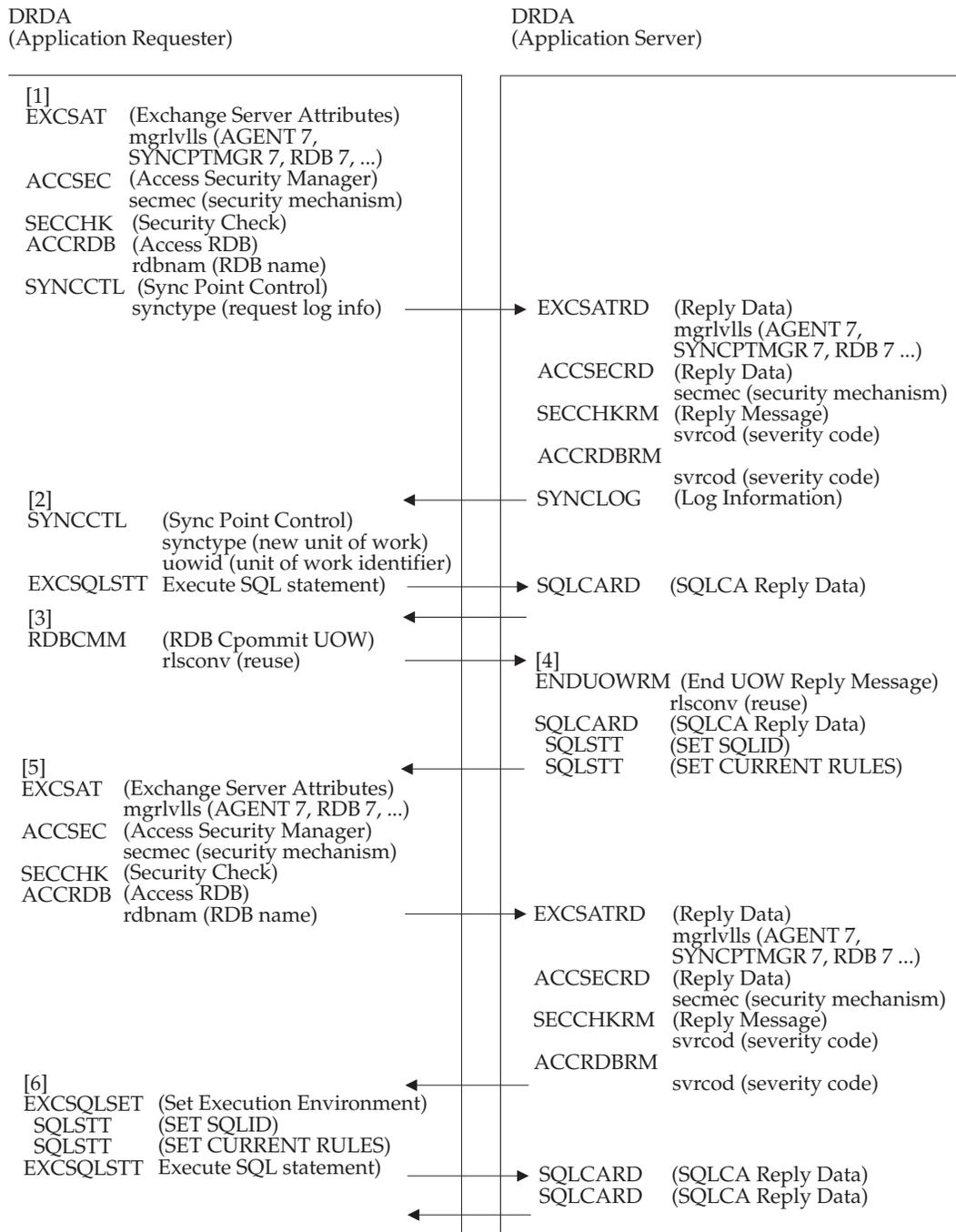
The following is a brief description of some of the parameters for the DDM commands that flow on a protected connection. This example illustrates the flow using DDM chaining to minimize the network costs. The DDM Reference provides a detailed description of the parameters and chaining requirements. If using the CMNSYNCPT manager (SNA 2-phase commit) on a protected connection or using an unprotected connection, the requester can issue the SYNCCTL command with the *synctype* parameter set to none and the release connection parameter set to reuse immediately after a commit or rollback prior to any other commands to request connection reuse.

1. The application requests a connection. The application requester establishes a network connection (as described in [Chapter 12](#) and [Chapter 13](#) which describe Network Protocols) with the application server. The requester issues the commands to establish the connection, authenticate the user, and access the RDB. The EXCSAT requests AGENT manager Level 7, SYNCPTMGR Level 7, and RDB manager Level 7 to indicate support for transaction pooling. The application server replies indicating support for transaction pooling.
2. The application requester starts a transaction by sending an execute SQL statement for the connected application. The application server replies with the SQL communications area.
3. The application commits the transaction. The application requester initiates the first phase of commit. The Prepare to Commit request indicates the network connection is to be released for reuse at the end of commit. If the requester rolls back the unit of work, the rollback request needs to indicate the network connection is to be released for reuse.

**Note:** If the requester rolls back the unit of work, the rollback request needs to indicate the network connection is to be released for reuse.

4. If the sync control requests the connection be released for reuse, the server always returns the release connection indicator. If the transaction closed all cursors, closed all protected RDB resources, and did not establish any special execution environment associated with the application which may impact the execution of a transaction by another application, the server returns an indicator that the connection is available for reuse; otherwise, the release connection indicator is returned as false. The server also provides the SQL SET statements to be issued to establish the application environment when another transaction is executed for the same application on possibly another connection. The SET statements are used to set all special registers associated with the application.
5. The second phase of commit completes. At the end of commit processing, the server must close and destroy all RDB resources associated with the server process related to the previous application.
6. When another application accesses the requester to execute a transaction, the requester checks for an existing available network connection. If an existing connection is available and has the same network requirements, the requester again issues the required commands to establish the connection, authenticate the user, access the RDB, and establish the application execution environment prior to executing the transaction.

**Pooling RUOW Network Connections**



**Figure 4-53** Reuse RUOW Connection using Transaction Pooling

The following is a brief description of some of the parameters for the DDM commands that flow on an unprotected connection. This example illustrates the flow using DDM chaining to minimize the network costs. The DDM Reference provides a detailed description of the parameters and chaining requirements.

1. The application requests a connection. The application requester establishes a network connection (as described in [Chapter 12](#) and [Chapter 13](#) which describe Network Protocols) with the application server. The requester issues the commands to establish the connection, authenticate the user, and access the RDB. The EXCSAT requests AGENT manager Level 7, and RDB manager Level 7 to indicate support for transaction pooling. The application server replies indicating support for transaction pooling.
2. The application requester starts a transaction by sending an execute SQL statement for the connected application. The application server replies with the SQL communications area.
3. The application commits the transaction. The RDB commit request indicates the network connection is to be released for reuse at the end of commit. If the requester rolls back the unit of work, the RDBRLLBCK request needs to indicate that the network connection is to be released for reuse.
4. If the RDB commit requests the connection be released for reuse, the server always returns the release connection indicator. If the transaction closed all cursors, closed all protected RDB resources, and did not establish any special execution environment associated with the application which may impact the execution of a transaction by another application, the server returns an indicator that the connection is available for reuse; otherwise, the release connection indicator is returned as false. The server also provides the SQL SET statements to be issued to establish the application environment when another transaction is executed for the application on another connection. At the end of commit processing, the server must close and destroy all RDB resources associated with the server process related to the previous application. The SET statements are used to set all special registers associated with the application.
5. When another application accesses the requester to execute a transaction, the requester checks for an existing available network connection. If an existing connection is available and has the same network requirements, the requester again issues the required commands to establish the connection, authenticate the user, access the RDB, and establish the application execution environment prior to executing the transaction.



## 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**. 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<sup>44</sup> or their abbreviations in describing data.

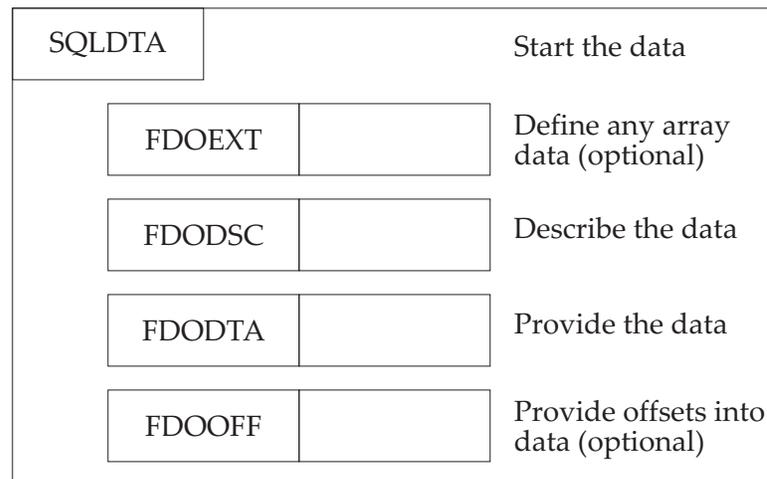
- MDD** Meta Data Definition
- SDA** Simple Data Array
- GDA** Group Data Array
- CPT** Continue Preceding Triplet
- RLO** Row LayOut

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 EXTDTA (Externalized Data), FDODSC (FD:OCA Descriptor), FDODTA (FD:OCA Data), FDOEXT (FD:OCA Extent Data), FDOOFF (FD:OCA Offset Data), OUTOVR (Output Override Descriptor), QRYDSC (Query Descriptor), and QRYDTA (Query Data). Descriptor objects are carriers for descriptors. The data objects are carriers for data. In commands where the descriptor and the data are available simultaneously (as is the case for

44. 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.

command data flowing to the relational database) the DDM command has a command data object (such as SQLDTA) that contains both the FDODSC (optionally the FDOEXT and FDOOFF objects) 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.

This FDODSC object describes all input, update, or parameter data for a single SQL statement. The descriptor carries type information by SDA references with zero extents for each input variable. The FDOEXT provides the extent specification for the input data when the input data contains a repeatable field. A repeatable field is used to describe an input array where each element in the array has the identical format, all having the same field length, field type, and type parameter. The SDA extents are implicitly specified in the FDOEXT carrier object and described by the SQLNUMEXT early descriptor. The FDOEXT contains the SDA extent specification and the FDOOFF contains the offset value for each field described in the FDODSC. If the input data does not contain any repeatable fields, then the FDOEXT and FDOOFF is not needed.

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 Generalized String indicator flag set on and the data object contains an FD:OCA Generalized String header, which is made up of a mode byte being X'02' and a length for the value. OUTOVR is a descriptor object that allows the application requester to control the format of output 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.

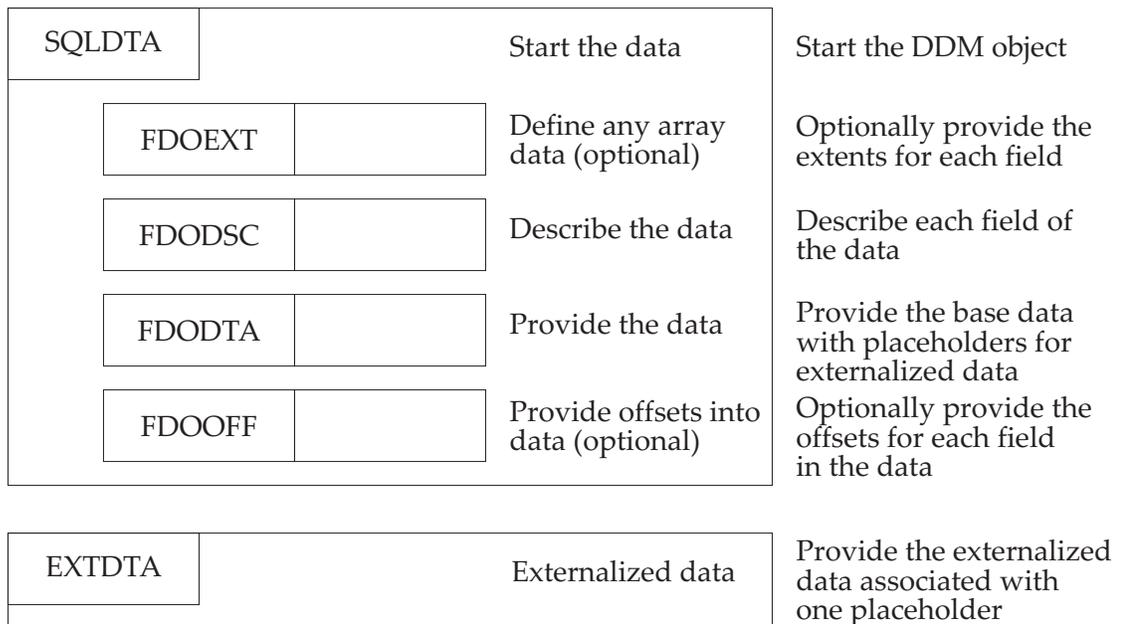
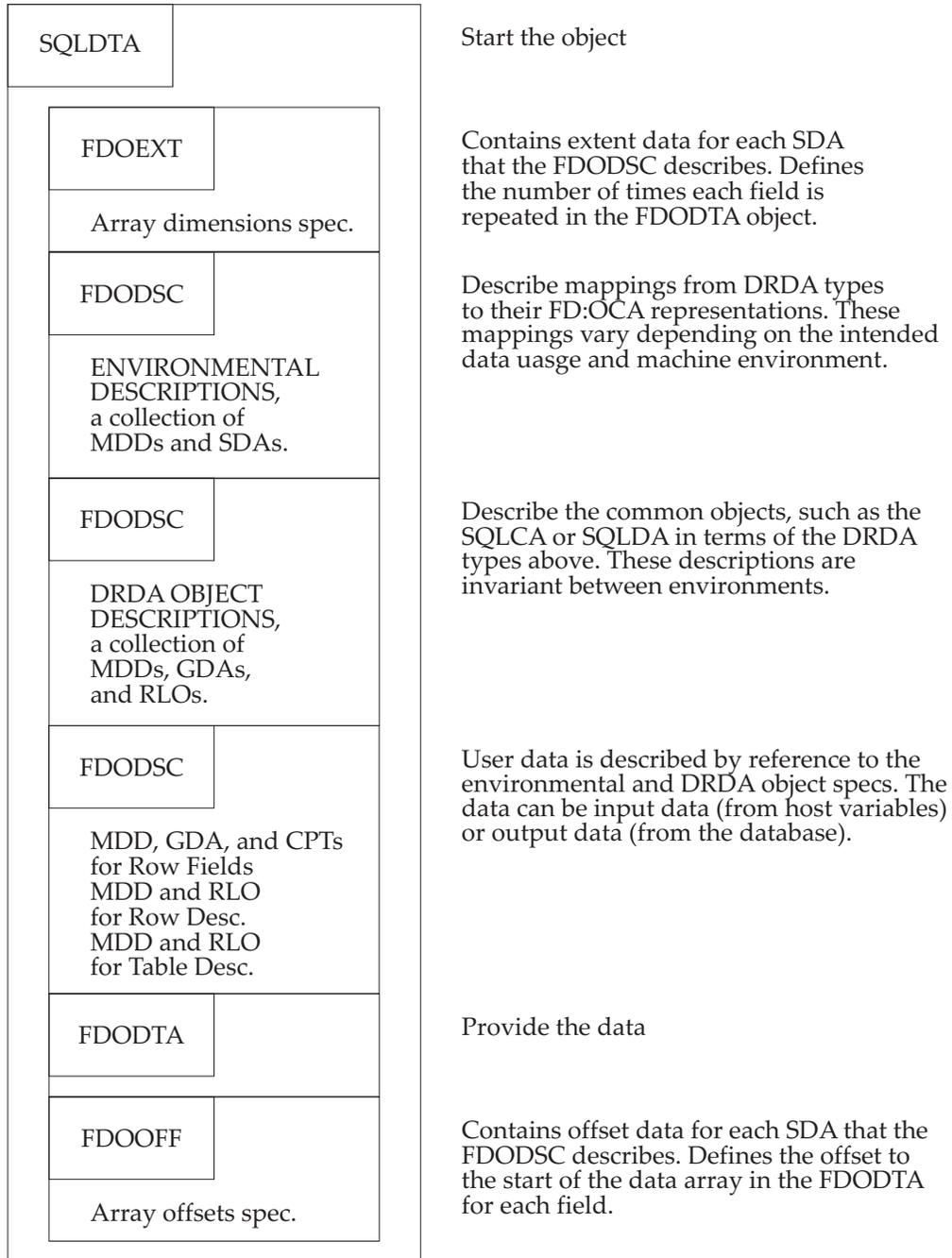


Figure 5-2 Basic FD:OCA Object with Externalized Data

**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.



**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 the DDM FDOEXT, FDODTA, and FDOOFF objects. 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.<sup>45</sup> DRDA defines an entire set of data types for each environment supported. See [Section 5.6.5](#) 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 first and last sections are optional. It provides the FD:OCA extent specifications when the input data contains a repeatable field. It also provides the relative offset in bytes to the data for each field from the start of the FDODTA. A repeatable field is used to describe an input array where each element in the array has the identical format, all having the same field length, field type, and type parameter. This FDODSC object describes all input, update, or parameter data for a single SQL statement. The descriptor carries type information by SDA reference with zero extents for each input variable. The SDA extents are implicitly specified in the FDOEXT carrier object and described by the SQLNUMEXT early descriptor. After the FDODTA carrier, the offset value for each field described in the FDODSC is specified in the FDOOFF carrier object. If the input data does not contain any repeatable fields, then the FDOEXT and FDOOFF are not needed.

The DDM FDODTA object 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](#) for more detail.

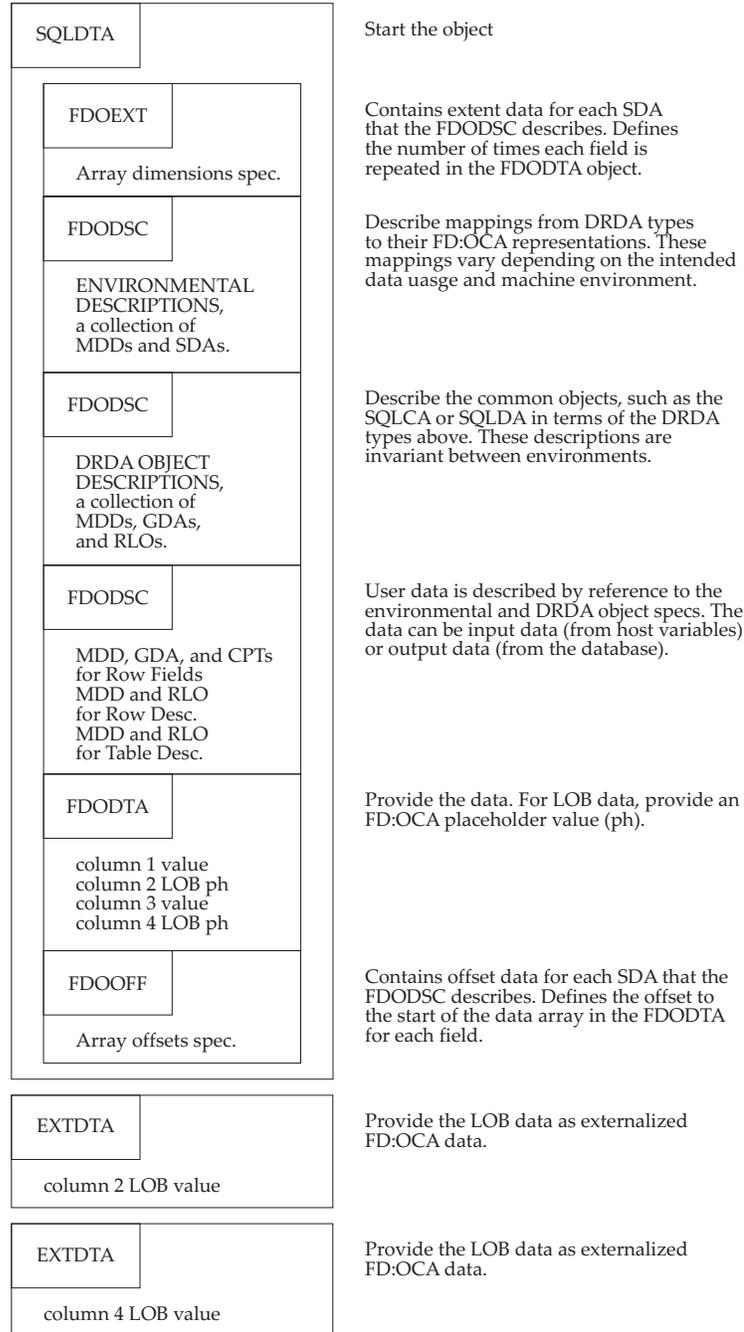
The organization of the FD:OCA descriptive triplets as shown in [Figure 5-3](#) gives the benefits of environment-independent specification of user data and commonly used information blocks.

---

45. 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 basic floating point are not. DRDA also has types for common collections of fields where SQL does not.

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 externalized input LOB data is given below. It includes the definition and flow of externalized FD:OCA data. [Figure 5-4](#) shows this usage.



**Figure 5-4** Conceptual View of a DRDA FD:OCA Object with Externalized 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 Generalized String header values in FDOTA. The actual externalized LOB data values are carried in the EXTDTA object in the order in which they appear in the FDOTA 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 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 and the sections following the figure define these descriptors. These are agreed to at EXCSAT time by means of the MGRLVL parameter for the SQL Application Manager, SQLAM X'2407'. (See the DDM Reference for definitions of these variables.) At an intermediate server, additional manager-level control is obtained through the use of the MGRLVLOVR object. See Section 4.3.5 (on page 85).

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 codepoint 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 codepoint.
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 codepoint 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 codepoint. This case includes all commands in Table 5-1 (on page 256), 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 codepoint 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 externalized values in the input row or answer set, the descriptors indicate whether an FD:OCA Generalized String header will flow for a column, and if so, each externalized 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 FD:OCA Generalized Strings in the answer set when the application server processes the OPNQRY command, the QRYDSC indicates that an FD:OCA Generalized String header will flow for each FD:OCA Generalized String column, and the placement of the value portion depends on the mode. If there is no LOB data in the answer set or OUTOVR is set to OUTOVRNON, Dynamic Data Format is enabled and QRYPRCTYP is LMTBLKPRC, then the application server sends a QRYDTA object and its associated EXTDTA objects after OPNQRYRM. The actual data format used to represent the FD:OCA Generalized String data in QRYDTA is determined by the application server according to the last OUTOVR if any and the Data Format (DF Rules); see [Section 7.8](#) for more details.

- 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 FD:OCA Generalized Strings in any result set when the stored procedure is executed, the QRYDSC indicates that an FD:OCA Generalized String header will flow for each FD:OCA Generalized String column. If there is no LOB data in the result set or OUTOVR is set to OUTOVRNON, Dynamic Data Format is enabled and QRYPRCTYP is LMTBLKPRC, then the application server sends a QRYDTA object and its associated EXTDTA objects after each OPNQRYRM. The actual data format used to represent the FD:OCA Generalized String data in QRYDTA is determined by the application server according to the last OUTOVR if any and the Data Format (DF Rules); see [Section 7.8](#) for more details.

- 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, SQLRSLRD, SQLCINRD, SQLSTT, SQLSTTVRB, SQLOBJNAM, FDOEXT, and FDOOFF, 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. The FDOEXT is required if the SQLDTA contains an input array. An FDODSC, an FDODTA, and an optional FDOEXT and FDOOFF are contained within SQLDTA. An FDODSC and an FDODTA are contained within the SQLDTARD. When a QRYDSC or a QRYDTA is used, one of each is all that is logically required. Also, the 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 shows data associated with each DRDA command described in Section 4.3.1.11 (on page 74). All descriptors named here are described later in this chapter.

Table 5-1 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 codepoint 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 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

DRDA Command	Command or Reply Data	DDM Object Name	DRDA Descriptor Name	Data Content Description
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
CLSQR	Command	None.	None.	None.
CLSQR	Reply	SQLCARD	SQLCARD	Return Code/Status
CNTQR (Note 2)	Command	OUTOVR	SQLDTA	Output Override Descriptor
CNTQR (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
DRPPKG	Command	None.	None.	None.
DRPPKG	Reply	SQLCARD	SQLCARD	Return Code/Status
DSCRDBTBL	Command	SQLOBJNAM	SQLOBJNAM	SQL Table Name
DSCRDBTBL	Reply	SQLCARD or SQLDARD	SQLCARD or SQLDARD	Return Code/Status Table Description
DSCSQLSTT	Command	None.	None.	None.
DSCSQLSTT	Reply	SQLCARD or SQLDARD	SQLCARD or SQLDARD	Return Code/Status Result Row or Input Parameter Description Including Labels
ENDBND	Command	None.	None.	None.
ENDBND	Reply	SQLCARD	SQLCARD	Return Code/Status
EXCSQLIMM	Command	SQLSTT	SQLSTT	SQL Statement (No Variable References)
EXCSQLIMM	Reply	SQLCARD	SQLCARD	Return Code/Status
EXCSQLSET	Command	None.	None.	None.
EXCSQLSET	Reply	SQLCARD	SQLCARD	Return Code/Status
EXCSQLSTT (Notes 1, 2)	Command	SQLDTA or OUTOVR and SQLDTA  or SQLDTA  and EXTDTA or OUTOVR and SQLDTA and EXTDTA	SQLDTA or SQLDTAMRW SQLDTA SQLDTA or SQLDTAMRW SQLDTA or SQLDTAMRW  SQLDTA SQLDTA or SQLDTAMRW	Data Descriptors and Values Data Descriptors and Values
EXCSQLSTT (Notes 2,3,4,5)	Reply	SQLCARD and SQLRSLRD and SQLCINRD	SQLCARD and SQLRSLRD and SQLCINRD	Return Code/Status  Information about Result Sets  Column Information for a Result Set

DRDA Command	Command or Reply Data	DDM Object Name	DRDA Descriptor Name	Data Content Description
		and QRYDSC and QRYDTA or SQLDTARD and SQLRSLRD and SQLCINRD and QRYDSC and QRYDTA or SQLDTARD and SQLRSLRD and SQLCINRD and QRYDSC and EXTDTA or SQLDTARD or SQLDTARD and EXTDTA	and SQLDTARD or SQLDTARD and SQLRSLRD and SQLCINRD and SQLDTARD or SQLDTARD and SQLRSLRD and SQLCINRD and SQLDTARD or SQLDTARD or SQLDTARD	Reply Data Descriptor and Values Return Code/Status and Output Parameter Values Information about Result Sets Column Information for a Result Set Reply Data Descriptor and Values Return Code/Status and Output Parameter Values Information about Result Sets Column Information for a Result Set Reply Data Descriptor and Values Externalized FD:OCA Data Reply Data Descriptor and Values Reply Data Descriptor and Values Externalized FD:OCA Data
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
PRPSQLSTT	Command	SQLATTR and SQLSTT	SQLSTT	SQL Statement or Attributes (No Variable Reference)
PRPSQLSTT	Reply	SQLCARD or SQLDARD	SQLCARD or SQLDARD	Return Code/Status Result Row Description including Labels
RDBCMM	Command	None.	None.	None.
RDBCMM	Reply	SQLCARD	SQLCARD	Return Code/Status
RDBRLLBCK	Command	None.	None.	None.
RDBRLLBCK	Reply	SQLCARD	SQLCARD	Return Code/Status
REBIND	Command	None.	None.	None.
REBIND	Reply	SQLCARD	SQLCARD	Return Code/Status

**Notes:**

1. SQLDTAMRW is not supported in DRDA Level 1.
2. EXTDTA and OUTOVR for externalized LOB data 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 nor any of its associated EXTDTA objects is returned until the first CNTQRY for the result set, unless OUTOVROPT is set to OUTOVRNON, DYNDTAFMT is set to TRUE, and QRYPRCTYP is LMTBLKPRC.
6. If any LOB data is in the query, then no QRYDTA is returned until the first CNTQRY, unless OUTOVROPT is set to OUTOVRNON, DYNDTAFMT is set to TRUE, and QRYPRCTYP is LMTBLKPRC.

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 256). 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. **Environment 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 basic floating point numbers should be represented. The SDAs also represent integer data, such as byte reversed.

2. **Simple 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 for each referenced environmental descriptor for input and output data to optimize the amount of storage allocated. The default SDA length for character data is 32,767, which allows an override up to that value. For an SQLDTAGRP late group descriptor, the length override can sometimes be zero (X'0000'), as detailed in [Section 5.5.3.1](#) (on page 271).

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.)

5. **Complex group descriptors** contain one or more complex fields to form a collection or a group of fields. A complex group may consist of one or more environmental descriptors, simple group descriptors, other complex group descriptors, and/or array descriptors. A complex field is an array descriptor that instantiates a field as an open-ended structure within the group. Each array in a group consists of a row which provides the number of rows in the array. Nullable fields within the group provide an indicator that indicates the presence or absence of the whole field. The triplets for such a descriptor can be Environmental Descriptor SDAs (in which case the overrides are applied), Group Descriptor GDAs, and other Array Descriptor RLOs. No overrides can occur for GDAs and RLOs. The referenced GDAs and RLOs can be either nullable or non-nullable. Nullability of a field is transmitted as a single byte. Array descriptors in a complex group consist of a row. A row consists of other groups. A group can contain simple groups and complex groups. A simple group consists of fields made up of other groups or simple data arrays. A complex group consists of fields made up of one or more arrays and other groups or simple data arrays.

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 401).

The early descriptors never actually flow on the link. By default, the Environmental Descriptors are determined by ACCRDB processing by means of TYPDEFNAM and TYPDEFOVR. By default, the Group, Row, and Array descriptors are agreed during EXCSAT processing by means of the MGRLVL parameter for the SQL application manager. The default TYPDEFNAM and/or TYPDEFOVR values can be overridden. See [Section 7.9](#) (on page 448). At an intermediate server, additional manager-level control is obtained through the use of the MGRLVLOVR object. See [Section 4.3.5](#) (on page 85).

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 427).)

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 alphabets, 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 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 266). 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](#) and [Figure 5-6](#) (on page 265). 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 SQLDTARD: SQL Communication Area with Data Array Description

**SQLDTARD Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'04'	X'01'	X'F0'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
6	X'71'	X'F0'

**SQLDTARD Data**

		byte 10	byte 11	byte 12
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Row LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLCADTA	X'01'	X'E0'	0 (all)	0 (all)

Descriptor in HEX:077800050401F0 0671F0 E00000

**Figure 5-5** SQLDTARD Array Descriptor

These are the descriptions for the Late Array Descriptor parameters shown in [Figure 5-5](#) (on page 264).

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 Length of the FD:OCA Meta Data Definition (MDD) triplet is always 7 for DRDA.
- Byte 1 MDD type indicator is always X'78' for MDDs.
- Byte 2 Identity identified by the Local Identifier (LID) 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 387).
- Byte 5 Meta Data Reference type for the MDD. The type identifies that the next byte identifies a late or early descriptor. The type is an X'01' if the next byte references a DRDA late descriptor. The type is an X'02' if the next byte references DRDA early Descriptors. Environmental Descriptors are defined as an early descriptor but can be overridden as a late descriptor.
- Byte 6 Meta Data Reference value for the MDD. DRDA uses this value as the DRDA Type indicator. [Table 5-11](#) describes the acceptable values.
- Byte 7 Length of the data triplets for the MDD.
- Byte 8 Data Definition type indicator for the MDD.
- Byte 9 Identify the local identifier (LID) for the referenced DRDA data type for the MDD.

5.5.1.2 *SQLDTAMRW: Data Array Description for Multi-Row Input***SQLDTAMRW Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'04'	X'01'	X'F4'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
6	X'71'	X'F4'

**SQLDTAMRW Data**

		byte 10	byte 11	byte 12
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Row LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLDTA	X'01'	X'E4'	0 (all)	0 (all)

Descriptor in HEX:077800050401F4 0671F4 E40000

**Figure 5-6** SQLDTAMRW Array Descriptor (Multi-Row Input 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 264)). However, in the middle there is one RLO triplet that refers to one or more group descriptors (GDAs) described in [Section 5.5.3](#) and in [Section 5.6.4](#) (on page 296).

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 SQLDTA: Data Description for One Row of Data

**SQLDTA Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'03'	X'01'	X'E4'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
6	X'71'	X'E4'

**SQLDTA Data**

		byte 10	byte 11	byte 12
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Group LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLDTAGRP	X'01'	X'D0'	0 (all)	1

Descriptor in HEX:077800050301E4 0671E4 D00001

**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 SQLCADTA: Data Description for One Row with SQL Communication Area and Data

**SQLCADTA Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'03'	X'01'	X'E0'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
9	X'71'	X'E0'

**SQLCADTA Data**

		1 byte	1 byte	1 byte
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Group LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLCAGRP	X'02'	X'54'	0 (all)	1
SQLDTAGRP	X'01'	X'D0'	0 (all)	1

Descriptor in HEX:077800050301E0 0971E0 540001 D00001

**Figure 5-8** SQLCADTA Row Descriptor

### 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 264). 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 mode byte plus the length portion of the data. The first bit is always '1'b, which indicates that the DRDA object carrying the data described by the SQLDTAGRP contains an FD:OCA Generalized String header, which is made up of a mode byte and the length portion of the string, while the value portion of the string may be returned in QRYDTA or EXTDTA, or as a progressive reference depending on the mode byte value. See the Data Format (DF Rules) in [Section 7.8](#) for more details.

DRDA specifies a fixed-length override value of 9, indicating that the FD:OCA Generalized String header size is 9 bytes, which consists of a 1-byte mode and an 8-byte length.

An EXTDTA object must flow for each FD:OCA Generalized String described, except under the following conditions:

1. It can be determined at the time its FD:OCA Generalized String header is generated that the nullable data is null.
2. It can be determined at the time its FD:OCA Generalized String header is generated that the nullable or non-nullable data has a length of zero bytes.
3. The mode byte in the FD:OCA Generalized String header is X'01' or X'03'.

Therefore, a FD:OCA Generalized String must have an EXTDTA reply data object associated with it under all other conditions, which include but are not limited to the following special circumstances:

1. It cannot be determined at the time the FD:OCA Generalized String header is generated whether the corresponding nullable data is null or not.
2. The FD:OCA Generalized String header indicates an unknown length for the data because its length cannot be determined at that time.

For details, refer to EXTDTAOVR in the DDM Reference.

All of the numbers in the boxes are in decimal unless otherwise noted. See [Section 5.6.6](#) and [Section 5.6.5](#) 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) unless otherwise noted.

## 5.5.3.1 SQLDTAGRP: Data Descriptions for One Row of Data

**SQLDTAGRP Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'02'	X'01'	X'D0'

**SQLDTAGRP Data Descriptor and Column Data**

(Describe up to 84 columns.)

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
3n+3 where n<85	N-GDA X'76'	X'D0'

		1 byte	2 bytes
<i>Descriptor Label</i>	<i>Ref Type</i>	<i>Env LID</i>	<i>Length Override</i>
Column 1	X'01'	Data Type	Data Length Override
Column 2	X'01'	Data Type	Data Length Override
...			
Column n-1	X'01'	Data Type	Data Length Override
Column n where n<85	X'01'	Data Type	Data Length Override

Each column is described by this group up to a maximum of 84 columns.

*n* is the total number of columns described by the group of columns.If *n*>84, then the next group of columns is described using the continue preceding triplet (CPT) descriptor.**SQLDTAGRP Continue Data Descriptor**

(Additional groups required to describe every column.)

byte y	byte y+1	byte y+2
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
3n+3 where n<85	CPT X'7F'	X'00'

		byte 3(x-1)+(y+3) for 1 byte	byte 3(x-1)+(y+4) for 2 bytes
<i>Descriptor Label</i>	<i>Ref Type</i>	<i>Env LID</i>	<i>Length Override</i>
Column <i>x</i>	X'01'	Data Type	Data Length Override
Column <i>x</i> +1	X'01'	Data Type	Data Length Override
...			
Column <i>n</i> -1	X'01'	Data Type	Data Length Override
Column <i>n</i> where <i>n</i> <85	X'01'	Data Type	Data Length Override

*y* is the first byte of the continue preceding triplet (CPT).*x* is the next column to be described by the current group.*n* is the total number of columns described by the current group of columns up to a maximum of 84 columns.

As many continue preceding triplets (CPT) are used as needed to describe the total number of columns.

```
Descriptor in HEX:077800050201D0 ...76D0...
                        (1st group of column descriptions)
                        ...7F00...
                        (Continued group of column descriptions)
```

**Figure 5-9** SQLDTAGRP Group Descriptor

The dots for the descriptor in hex indicate that the values are not known until runtime.

The SQLDTAGRP descriptor indicates the corresponding FD:OCA object is nullable. For output data that is originating at a server, the null indicator is used to indicate for an error condition that the row is null.

For input data that is originating at an application requester, the null indicator must be zero or the server must reject it with a DTAMCHRM reply message, except in the case of a multi-row input operation where the following applies:

- A null indicator value of zero (X'00') indicates row data is present. This is not an error condition.
- Any other indicator value indicates the row is null (that is, there is no row data). Specifically:
  - A null indicator value of -1 (X'FF') indicates that the server should skip over the null row. This is not an error condition.
  - A null indicator value of -2 (X'FE') indicates that the server should return an error SQLCA containing SQLSTATE 22527.
  - A null indicator value of -3 (X'FD') indicates that the server should skip over the null row which is also the last row for a multi-row input operation. This is not an error condition. Therefore, if there are fewer rows than indicated previously by the NBRROW instance variable on the EXCSQLSTT command, a PRCCNVRM reply message should not be returned.
  - For all other non-zero (X'FC'-X'80', X'01'-X'7F') null indicator values, the server must return a DTAMCHRM reply message.
- Null rows count towards the total number of rows for a multi-row input operation as indicated by the NBRROW instance variable on the EXCSQLSTT command.
- Null rows are allowed for both atomic and non-atomic multi-row input operations.

The length override used for each column in the N-GDA or CPT triplet must not be zero (X'0000') for either input or output data except for the following types:

Application Class	Meta Data Type	Meta Data Reference DRDA-Type and Name	SQL Type	Description
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

Application Class	Meta Data Type	Meta Data Reference DRDA-Type and Name	SQL Type	Description
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
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

Note that any allowed length override of zero (X'0000') on input or output indicates the length of the column value is literally zero. This is not equivalent to the column value being null.

### 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.

#### LOB Externalization

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.
- If the OUTOVR command data object is flowed following an EXCSQLSTT command which is not a stored procedure call, and the number of triplets in the SQLDTAGRP FD:OCA descriptor does not match the number of columns in the select list, the following applies:
  - If the number of triplets in the OUTOVR SQLDTAGRP FD:OCA descriptor is greater than the number of columns in the select list, then the server ignores the outstanding trailing triplets in the OUTOVR SQLDTAGRP FD:OCA descriptor.
  - If the number of triplets in the OUTOVR SQLDTAGRP FD:OCA descriptor is fewer than the number of columns in the select list, then the server can either return only those columns in the SQLDTARD reply data object, or it can return the same number of columns as the select list by treating the missing triplets in the OUTOVR SQLDTAGRP FD:OCA descriptor as default triplets.

- If the OUTOVR command data object is flowed following a CNTQRY command, and the number of triplets in the SQLDTAGRP FD:OCA descriptor does not match the number of columns in the select list, the following applies:
  - If the number of triplets in the OUTOVR SQLDTAGRP FD:OCA descriptor is greater than the number of columns in the select list, then the server ignores the outstanding trailing triplets in the OUTOVR SQLDTAGRP FD:OCA descriptor.
  - If the number of triplets in the OUTOVR SQLDTAGRP FD:OCA descriptor is fewer than the number of columns in the select list, then the server must return the same number of columns as the select list by treating the missing triplets in the OUTOVR SQLDTAGRP FD:OCA descriptor as default triplets.
- The nullability of a LOB locator as returned by the server in an SQLDTARD or QRYDTA reply data object in response to an OUTOVR command data object must match that of the corresponding LOB value that is being overridden. As such, the nullability of a LOB locator override triplet as specified in the OUTOVR command data object that is associated with an EXCSQLSTT or CNTQRY command is ignored by the server.

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 NOCL 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), and 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.

A requester or server commits default 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 QTDSQLVX. Accepting one of these environments commits the requester or 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. The default TYPDEFNAM and/or TYPDEFOVR values can be overridden. See [Section 7.9](#) (on page 448). At an intermediate server, additional manager-level control is obtained through the use of the MGRLVLOVR object. See [Section 4.3.5](#) (on page 85).

### 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) and externally encoded XML data is defined by 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](#) describes 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: Data Array of a Result Set

**SQLRSLRD Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'04'	X'02'	X'7F'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
9	X'71'	X'7F'

**SQLRSLRD Data**

		1 byte	1 byte	1 byte
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Row LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLNUMROW	X'02'	X'68'	0 (all)	1
SQLRSROW	X'02'	X'6F'	0 (all)	0 (all)

Descriptor in HEX:0778000504027F 09717F 680001 6F0000

**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: SQL Result Set Column Array Description

**SQLCINRD Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'04'	X'02'	X'7B'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
12	X'71'	X'7B'

**SQLCINRD Data**

		1 byte	1 byte	1 byte
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Row LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLDHROW	X'02'	X'E0'	0 (all)	1
SQLNUMROW	X'02'	X'68'	0 (all)	1
SQLDAROW	X'02'	X'60'	0 (all)	0 (all)

Descriptor in HEX:0778000504027B 0C717B E00001 680001 6F0000

**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: SQL Statement Variable Description

**SQLSTTVRB Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'04'	X'02'	X'7E'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
9	X'71'	X'7E'

**SQLSTTVRB Data**

		1 byte	1 byte	1 byte
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Row LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLNUMROW	X'02'	X'68'	0 (all)	1
SQLVRBROW	X'02'	X'6E'	0 (all)	0 (all)

Descriptor in HEX:0778000504027E 09717E 680001 6E0000

**Figure 5-12** SQLSTTVRB Array Descriptor

The SQLSTTVRB Array Descriptor figure describes the variables appearing in an SQL statement.

5.6.2.4 SQLDARD: SQL Descriptor Area Row Description with SQL Communication Area

A requester can control how an SQLDARD object is generated. Refer to [Section 4.4.16](#) for more details on a light descriptor, a standard descriptor, or an extended descriptor. The SQLNUMROW object identifies the number of variables described. The number of SQLDAROWs must match the number in the SQLNUMROW object. The SQLDHGRP object provides statement-level descriptive information. The SQLDARD contains only one SQLDHGRP object.

**SQLDARD Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'04'	X'02'	X'74'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
15	X'71'	X'74'

**SQLDARD Data**

		1 byte	1 byte	1 byte
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Row LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLCARD	X'02'	X'64'	0 (all)	1
SQLDHROW	X'02'	X'E0'	0 (all)	1
SQLNUMROW	X'02'	X'68'	0 (all)	1
SQLDAROW	X'02'	X'60'	0 (all)	0 (all)

Descriptor in HEX:07780005040174 0F7174 640001 E00001 680001 600000

**Figure 5-13** SQLDARD Array Descriptor

5.6.2.5 SQLDCTOKS: SQL Diagnostics Condition Token Array

**SQLDCTOKS Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'04'	X'02'	X'F7'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
9	X'71'	X'F7'

**SQLDCTOKS Data**

		1 byte	1 byte	1 byte
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Row LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLNUMROW	X'02'	X'68'	0 (all)	1
SQLTOKROW	X'02'	X'E7'	0 (all)	0 (all)

Descriptor in HEX:077800050402F7 0971F7 680001 E70000

**Figure 5-14** SQLDCTOKS Array Descriptor

5.6.2.6 *SQLDIAGCI: SQL Diagnostics Condition Information Array***SQLDIAGCI Meta Data and Data Descriptor**

<b>byte 0</b>	<b>byte 1</b>	<b>byte 2</b>	<b>byte 3</b>	<b>byte 4</b>	<b>byte 5</b>	<b>byte 6</b>
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'04'	X'02'	X'F5'

<b>byte 7</b>	<b>byte 8</b>	<b>byte 9</b>
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
9	X'71'	X'F5'

**SQLDIAGCI Data**

		<b>1 byte</b>	<b>1 byte</b>	<b>1 byte</b>
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Row LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLNUMROW	X'02'	X'68'	0 (all)	1
SQLDCROW	X'02'	X'E5'	0 (all)	0 (all)

Descriptor in HEX:077800050402F5 0971F5 680001 E50000

**Figure 5-15** SQLDIAGCI Array Descriptor

5.6.2.7 SQLDIAGCN: SQL Diagnostics Connection Array

**SQLDIAGCN Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'04'	X'02'	X'F6'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
9	X'71'	X'F6'

**SQLDIAGCN Data**

		1 byte	1 byte	1 byte
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Row LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLNUMROW	X'02'	X'68'	0 (all)	1
SQLCNROW	X'02'	X'E6'	0 (all)	0 (all)

Descriptor in HEX:077800050402F6 0971F6 680001 E60000

**Figure 5-16** SQLDIAGCN Array Descriptor

### 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 SQLRSROW: SQL Row Description of One Result Set Row

##### SQLRSROW Meta Data and Data Descriptor

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'03'	X'02'	X'6F'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
6	X'71'	X'6F'

##### SQLRSROW Data

		byte 10	byte 11	byte 12
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Group LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLRSGRP	X'02'	X'5F'	0 (all)	1

Descriptor in HEX:0778000503026F 06716F 5F0001

Figure 5-17 SQLRSROW Row Descriptor

5.6.3.2 SQLVRBROW: SQL Statement Variable Row Description

**SQLVRBROW Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'03'	X'02'	X'6E'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
6	X'71'	X'6E'

**SQLVRBROW Data**

		byte 10	byte 11	byte 12
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Group LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLVRBGRP	X'02'	X'5E'	0 (all)	1

Descriptor in HEX:0778000503026E 06716E 5E0001

**Figure 5-18** SQLVRBROW Row Descriptor

## 5.6.3.3 SQLSTT: SQL Statement Row Description

**SQLSTT Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'03'	X'02'	X'6C'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
6	X'71'	X'6C'

**SQLSTT Data**

		byte 10	byte 11	byte 12
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Group LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLSTTGRP	X'02'	X'5C'	0 (all)	1

Descriptor in HEX:0778000503026C 06716C 5C0001

**Figure 5-19** SQLSTT Row Descriptor (One SQL Statement or SQL Attributes)

This row descriptor consists of a single long variable character string that contains a full SQL statement or the SQL attributes. 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 attributes should have leading and trailing blanks removed.

The binder must specially treat SQL statements that contain references to program variables. Detailed rules are listed in [Section 7.12](#) (on page 454).

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.

5.6.3.4 SQLOBJNAM: SQL Object Name Row Description

**SQLOBJNAM Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'03'	X'02'	X'6A'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
6	X'71'	X'6A'

**SQLOBJNAM Data**

		byte 10	byte 11	byte 12
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Group LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLOBJGRP	X'02'	X'5A'	0 (all)	1

Descriptor in HEX:0778000503026A 06716A 5A0001

**Figure 5-20** SQLOBJNAM Row Descriptor

## 5.6.3.5 SQLNUMROW: SQL Number of Elements Row Description

**SQLNUMROW Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'03'	X'02'	X'68'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
6	X'71'	X'68'

**SQLNUMROW Data**

		byte 10	byte 11	byte 12
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Group LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLNUMGRP	X'02'	X'58'	0 (all)	1

Descriptor in HEX:07780005030268 067168 580001

**Figure 5-21** SQLNUMROW Row Descriptor

5.6.3.6 SQLCARD: SQL Communication Area Row Description

**SQLCARD Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'03'	X'02'	X'64'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
6	X'71'	X'64'

**SQLCARD Data**

		byte 10	byte 11	byte 12
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Group LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLCAGRP	X'02'	X'54'	0 (all)	1

Descriptor in HEX:07780005030264 067164 540001

**Figure 5-22** SQLCARD Row Descriptor

## 5.6.3.7 SQLDAROW: SQL Descriptor Area Row Description

**SQLDAROW Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'03'	X'02'	X'60'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
6	X'71'	X'60'

**SQLDAROW Data**

		byte 10	byte 11	byte 12
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Group LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLDAGRP	X'02'	X'50'	0 (all)	1

Descriptor in HEX:07780005030260 067160 500001

**Figure 5-23** SQLDAROW Row Descriptor

5.6.3.8 *SQLDHROW: SQL Descriptor Header Row Description***SQLDHROW Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'03'	X'02'	X'E0'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
6	X'71'	X'E0'

**SQLDHROW Data**

		byte 10	byte 11	byte 12
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Group LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLDHGRP	X'02'	X'D0'	0 (all)	1

Descriptor in HEX:077800050302E0 0671E0 D00001

**Figure 5-24** SQLDHROW Row Descriptor

## 5.6.3.9 SQLCNROW: SQL Diagnostics Connection Row

**SQLCNROW Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'03'	X'02'	X'E6'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
9	X'71'	X'E6'

**SQLCNROW Data**

		byte 10	byte 11	byte 12
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Group LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLCNGRP	X'02'	X'D6'	0 (all)	1

Descriptor in HEX:077800050302E6 0671E6 D60001

**Figure 5-25** SQLCNROW Row Descriptor

5.6.3.10 SQLDCROW: SQL Diagnostics Condition Row

**SQLDCROW Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'03'	X'02'	X'E5'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
6	X'71'	X'E5'

**SQLDCROW Data**

		byte 10	byte 11	byte 12
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Group LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLDCGRP	X'02'	X'D5'	0 (all)	1

Descriptor in HEX:077800050302E5 0671E5 D50001

**Figure 5-26** SQLDCROW Row Descriptor

## 5.6.3.11 SQLTOKROW: SQL Diagnostics Token Row

**SQLTOKROW Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'03'	X'02'	X'E7'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
6	X'71'	X'E7'

**SQLTOKROW Data**

		byte 10	byte 11	byte 12
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Group LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLTOKGRP	X'02'	X'D7'	0 (all)	1

Descriptor in HEX:077800050302E7 0671E7 D70001

**Figure 5-27** SQLTOKROW Row Descriptor

**5.6.4 Early Group Descriptors**

Figure 5-28 to Figure 5-36 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 *SQLRSGRP: SQL Result Set Group Description*

**SQLRSGRP Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'02'	X'02'	X'5F'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
15	X'75'	X'5F'

**SQLRSGRP Data**

			1 byte	1 byte
<i>Descriptor Label</i>	<i>DRDA Type</i>	<i>Ref Type</i>	<i>Env LID</i>	<i>Length Override</i>
SQLRSLOCATOR	RSL	X'02'	X'14'	4
SQLRSNAME_m	VCM	X'02'	X'3E'	255
SQLRSNAME_s	VCS	X'02'	X'32'	255
SQLRSNUMROWS	I4	X'02'	X'02'	4

Descriptor in HEX:0778000502025F 0F755F 140004 3E00FF 3200FF 020004

**Figure 5-28 SQLRSGRP Group Descriptor**

Table 5-4 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 SQLVRBGRP: SQL Statement Variable Group Description

**SQLVRBGRP Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'02'	X'02'	X'5E'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
33	GDA X'75'	X'5E'

**SQLVRBGRP Data**

			1 byte	2 bytes
<i>Descriptor Label</i>	<i>DRDA Type</i>	<i>Ref Type</i>	<i>Env LID</i>	<i>Length Override</i>
SQLPRECISION	I2	X'02'	X'04'	2
SQLSCALE	I2	X'02'	X'04'	2
SQLLENGTH	I8	X'02'	X'16'	8
SQLTYPE	I2	X'02'	X'04'	2
SQLCCSID	FB	X'02'	X'26'	2
SQLNAME_m	VCM	X'02'	X'3E'	255
SQLNAME_s	VCS	X'02'	X'32'	255
SQLDIAGNAME_m	VCM	X'02'	X'3E'	255
SQLDIAGNAME_s	VCS	X'02'	X'32'	255
SQLUDTGRP	N-GDA	X'02'	X'5B'	0

Descriptor in HEX:0778000502025E 21755E 040002 040002 160008 040002  
 260002 3E00FF 3200FF 3E00FF 3200FF  
 5B0000

The abbreviations *\_m* and *\_s* stand for mixed and single, respectively.

**Figure 5-29** SQLVRBGRP Group Descriptor

[Table 5-5](#) describes the usage of each of the fields of the DRDA SQL Data Area.

**Table 5-5** 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.

## 5.6.4.3 SQLSTTGRP: SQL Statement or Attributes Group Description

**SQLSTTGRP Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'02'	X'02'	X'5C'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
9	GDA X'75'	X'5C'

**SQLSTTGRP Data**

			1 byte	2 bytes
<i>Descriptor Label</i>	<i>DRDA Type</i>	<i>Ref Type</i>	<i>Env LID</i>	<i>Length Override</i>
SQLSTATEMENT_m	NOCM	X'02'	X'CF'	4
SQLSTATEMENT_s	NOCS	X'02'	X'CB'	4

Descriptor in HEX:0778000502025C 09755C CF0004 CB0004

**Figure 5-30** SQLSTTGRP Group Descriptor (One SQL Statement or Attributes)

This group defines a pair of variable character strings, one of which contains an SQL statement or SQL attributes. 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.

5.6.4.4 *SQLOBJGRP: SQL Object Name Group Description*

**SQLOBJGRP Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'02'	X'02'	X'5A'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
9	GDA X'75'	X'5A'

**SQLOBJGRP Data**

			1 byte	2 bytes
<i>Descriptor Label</i>	<i>DRDA Type</i>	<i>Ref Type</i>	<i>Env LID</i>	<i>Length Override</i>
SQLOBJECTNAME_m	VCM	X'02'	X'3E'	255
SQLOBJECTNAME_s	VCS	X'02'	X'32'	255

Descriptor in HEX:0778000502025A 09755A 3E00FF 3200FF

**Figure 5-31** SQLOBJGRP Group Descriptor

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.

## 5.6.4.5 SQLNUMGRP: SQL Number of Elements Group Description

**SQLNUMGRP Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'02'	X'02'	X'58'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
6	GDA X'75'	X'58'

**SQLNUMGRP Data**

			1 byte	2 bytes
<i>Descriptor Label</i>	<i>DRDA Type</i>	<i>Ref Type</i>	<i>Env LID</i>	<i>Length Override</i>
SQLNUM	I2	X'02'	X'04'	2

Descriptor in HEX:07780005020258 067558 040002

**Figure 5-32** 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.

5.6.4.6 SQLCAGRP: SQL Communication Area Group Description

**SQLCAGRP Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'02'	X'02'	X'54'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
18	N-GDA X'76'	X'54'

**SQLCAGRP Data**

			1 byte	2 bytes
<i>Descriptor Label</i>	<i>DRDA Type</i>	<i>Ref Type</i>	<i>Env LID</i>	<i>Length Override</i>
SQLCODE	I4	X'02'	X'02'	4
SQLSTATE	FCS	X'02'	X'30'	5
SQLERRPROC	FCS	X'02'	X'30'	8
SQLCAXGRP	N-GDA	X'02'	X'52'	0
SQLDIAGGRP	N-GDA	X'02'	X'D1'	0

Descriptor in HEX:07780005020254 127654 020004 300005 300008 520000 D10000

**Figure 5-33** SQLCAGRP Group Descriptor

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 503).

## 5.6.4.7 SQLCAXGRP: SQL Communication Area Exceptions Group Description

**SQLCAXGRP Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'02'	X'02'	X'52'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
63	N-GDA X'76'	X'52'

**SQLCAXGRP Data**

			1 byte	2 bytes
<i>Descriptor Label</i>	<i>DRDA Type</i>	<i>Ref Type</i>	<i>Env LID</i>	<i>Length Override</i>
SQLERRD1	I4	X'02'	X'02'	4
SQLERRD2	I4	X'02'	X'02'	4
SQLERRD3	I4	X'02'	X'02'	4
SQLERRD4	I4	X'02'	X'02'	4
SQLERRD5	I4	X'02'	X'02'	4
SQLERRD6	I4	X'02'	X'02'	4
SQLWARN0	FCS	X'02'	X'30'	1
SQLWARN1	FCS	X'02'	X'30'	1
SQLWARN2	FCS	X'02'	X'30'	1
SQLWARN3	FCS	X'02'	X'30'	1
SQLWARN4	FCS	X'02'	X'30'	1
SQLWARN5	FCS	X'02'	X'30'	1
SQLWARN6	FCS	X'02'	X'30'	1
SQLWARN7	FCS	X'02'	X'30'	1
SQLWARN8	FCS	X'02'	X'30'	1
SQLWARN9	FCS	X'02'	X'30'	1
SQLWARNA	FCS	X'02'	X'30'	1
SQLRDBNAME	VCS	X'02'	X'32'	255
SQLERRMSG_m	VCM	X'02'	X'3E'	70
SQLERRMSG_s	VCS	X'02'	X'32'	70

```
Descriptor in HEX:07780005020252 3F7652 020004 020004 020004 020004
020004 020004 300001 300001 300001
300001 300001 300001 300001 300001
300001 300001 300001 3200FF 3E0046
320046
```

**Figure 5-34** SQLCAXGRP Group Descriptor

SQL and individual implementations define the semantics of the values in each of the fields in [Figure 5-34](#) (on page 303). 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.

## 5.6.4.8 SQLDIAGGRP: SQL Diagnostics Group Description

**SQLDIAGGRP Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'02'	X'05'	X'D1'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
12	N-GDA X'76'	X'D1'

**SQLDIAGGRP Data**

			1 byte	2 bytes
<i>Descriptor Label</i>	<i>DRDA Type</i>	<i>Ref Type</i>	<i>Env LID</i>	<i>Length Override</i>
SQLDIAGSTT	N-GDA	X'52'	X'D3'	0
SQLDIAGCI	N-RLO	X'72'	X'F5'	0
SQLDIAGCN	N-RLO	X'72'	X'F6'	0

Descriptor in HEX:077800050202D1 0C76D1 D30000 F50000 F60000

**Figure 5-35** SQLDIAGGRP Group Descriptor

## 5.6.4.9 SQLDAGRP: SQL Descriptor Area Group Description

**SQLDAGRP Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'02'	X'02'	X'50'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
21	GDA X'75'	X'50'

**SQLDAGRP Data**

			1 byte	2 bytes
<i>Descriptor Label</i>	<i>DRDA Type</i>	<i>Ref Type</i>	<i>Env LID</i>	<i>Length Override</i>
SQLPRECISION	I2	X'02'	X'04'	2
SQLSCALE	I2	X'02'	X'04'	2
SQLLENGTH	I8	X'02'	X'16'	8
SQLTYPE	I2	X'02'	X'04'	2
SQLCCSID	FB	X'02'	X'26'	2
SQLDOPTGRP	N-GDA	X'02'	X'D2'	0

Descriptor in HEX:07780005020250 157550 040002 040002 160008 040002  
260002 D20000

**Figure 5-36** SQLDAGRP Group Descriptor

A requester can specify which SQLDAGRP group fields are to be returned by the server. The fields to be returned are identified by the DDM TYPSQLDA instance variable or the DDM RSLSETFLG on the SQL command. A light, standard, or extended SQLDAGRP can be requested.

- A light descriptor is when the SQLDOPTGRP returned is null.
- A standard descriptor is when the SQLDOPTGRP is returned without the SQLDXGRP. The SQLUDTGRP group is returned if the variable or column described is a user-defined type. The SQLDXGRP group is not returned and must be null.
- An extended descriptor is when the SQLDOPTGRP is returned with the SQLDXGRP group. The SQLUDTGRP group is returned if the variable or column being described is a user-defined data type. The SQLDXGRP group must be returned and should not be null.

Table 5-6 describes the usage of each of the fields of the DRDA SQL Data Area.

**Table 5-6** 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 5-6](#) (on page 307). 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.

5.6.4.10 SQLUDTGRP: SQL Descriptor User-Defined Type Group Description

**SQLUDTGRP Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'02'	X'02'	X'51'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
21	N-GDA X'76'	X'51'

**SQLUDTGRP Data**

			1 byte	2 bytes
<i>Descriptor Label</i>	<i>DRDA Type</i>	<i>Ref Type</i>	<i>Env LID</i>	<i>Length Override</i>
SQLUDTXTYPE	I4	X'02'	X'02'	4
SQLUDTRDB	VCS	X'02'	X'32'	255
SQLUDTSHEMA_m	VCM	X'02'	X'3E'	255
SQLUDTSHEMA_s	VCS	X'02'	X'32'	255
SQLUDTNAME_m	VCM	X'02'	X'3E'	255
SQLUDTNAME_s	VCS	X'02'	X'32'	255

Descriptor in HEX:07780005020251 157651 020004 3200FF 3E00FF 3200FF  
3E00FF 3200FF

**Figure 5-37 SQLUDTGRP Group Descriptor**

This group defines pairs of variable-length character strings. SQLUDTNAME\_m and SQLUDTNAME\_s are mutually-exclusive; SQLUDTSHEMA\_m and SQLUDTSHEMA\_s are mutually-exclusive; that is, only one with a non-zero length can be specified for the duplicated field. If both fields have a non-zero length, return an FD:OCA data mismatch reply message. (Refer to the DTAMCHRM term in the DDM Reference.) If both the SQLUDTSHEMA fields have a zero length, the value for the field is defaulted to the value in the SQLDHSHEMA field in the SQL Descriptor Header Group (SQLDHGRP) if present. If the SQLUDTRDB has a zero length, the value for the field is the value in the SQLRDBNAM field in the SQL Descriptor Header Group (SQLDHGRP) if present; otherwise, the default RDB name is the RDBNAM used to access the RDB.

Field Name	Usage
SQLUDTXTYPE	This field contains the user-defined type code for the field. The semantics of the type code are as follows: 0 means not a UDT; 1 means distinct type; 2 means structured type; and 3 means reference type. All other values are invalid or undefined.

Field Name	Usage
SQLUDTRDB	This field contains the default RDB name to which this SQL user-defined data type is defined. This is the high-level qualifier for this user-defined data type. The RDB name uniquely identifies the catalog associated with the SQL user-defined data type. If the SQLUDTRDB has a zero length, the SQLDRDBNAM identifies the value for this field if the SQLDHDRGRP is present; otherwise, the default RDB is the RDBNAM used to access the RDB.
SQLUDTSHEMA	This field contains the schema name to which this SQL user-defined data type is defined. This is the second-level qualifier for this user-defined data type. The SQLUDTSHEMA_m or SQLUDTSHEMA_s are mutually-exclusive. If both SQLUDTSHEMA_m or SQLUDTSHEMA_s have a zero length, the SQLDSHEMA identifies the value for this field if the SQLDHDRGRP is present.
SQLUDTNAME	This field contains the name of the user-defined data type. This is the unqualified name of this user-defined data type.

5.6.4.11 SQLDHGRP: SQL Descriptor Header Group Description

**SQLDHGRP Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'02'	X'02'	X'D0'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
30	N-GDA X'76'	X'D0'

**SQLDHGRP Data**

			1 byte	2 bytes
<i>Descriptor Label</i>	<i>DRDA Type</i>	<i>Ref Type</i>	<i>Env LID</i>	<i>Length Override</i>
SQLDHOLD	I2	X'02'	X'04'	2
SQLDRETURN	I2	X'02'	X'04'	2
SQLDSCROLL	I2	X'02'	X'04'	2
SQLDSENSITIVE	I2	X'02'	X'04'	2
SQLDFCODE	I2	X'02'	X'04'	2
SQLDKEYTYPE	I2	X'02'	X'04'	2
SQLDRDBNAM	VCS	X'02'	X'32'	255
SQLDSHEMA_m	VCM	X'02'	X'3E'	255
SQLDSHEMA_s	VCS	X'02'	X'32'	255

Descriptor in HEX:077800050202D0 1E76D0 040002 040002 040002 040002  
 040002 040002 3200FF 3E00FF 3200FF

**Figure 5-38 SQLDHGRP Group Descriptor**

The SQL Descriptor Header provides statement-level descriptive information. A requester specifies when a non-null SQL Descriptor Header is to be returned in the SQLDARD reply data. The DDM TYPSQLDA instance variable on the SQL command specifies when this group is returned. If the requester requests an extended descriptor, the SQLDHGRP must be returned. The group must not be null; otherwise, the group must be specified as null. If the server does not return the correct level of SQLDARD, an FD:OCA data mismatch should be generated. (Refer to the DTAMCHRM term in the DDM Reference.)

Field Name	Usage
SQLDHOLD	This field can have a value of 0, 1, or -1. A value of 1 indicates this statement is related to a cursor which is defined using the WITH HOLD clause. If the cursor is known to be defined to be without the WITH HOLD clause, the value is 0. If unknown, a value of -1 should be set.

Field Name	Usage
SQLDRETURN	This field can have a value of 0, 1, 2, or -1. A value of 1 indicates this statement is related to a cursor which is defined using the WITH RETURN CLIENT clause. A value of 2 indicates that the cursor is defined using the WITH RETURN CALLER clause. If the statement is not a query, the value is 0. If the statement has a cursor, but it is not known at the time of the DESCRIBE if the cursor is intended to be used as a result set that will be returned from a procedure, a value of -1 should be set.
SQLDSCROLL	This field can have a value of 0, 1, or -1. A value of 1 indicates this statement is related to a cursor which is defined using the SCROLL clause. If the related cursor is not scrollable, or no cursor exists for the statement, the value is 0. If the scrollability is not known at the time of the DESCRIBE, the value is -1.
SQLDSENSITIVE	<p>This field can have the following values:</p> <ul style="list-style-type: none"> <li>-1 The statement is related to a cursor, but the sensitivity is unknown at the time of the DESCRIBE operation.</li> <li>0 The statement is not related to a cursor and the SQLDSCROLL field has a value of 0.</li> <li>1 The statement is related to a cursor which is defined as SENSITIVE DYNAMIC.</li> <li>2 The statement is related to a cursor which is defined as SENSITIVE STATIC.</li> <li>3 The statement is related to a cursor which is defined as INSENSITIVE.</li> <li>4 The statement is related to a cursor which is defined with PARTIAL SENSITIVITY and STATIC size and ordering.</li> <li>5 The statement is related to a cursor which is defined with PARTIAL SENSITIVITY and DYNAMIC size and ordering.</li> </ul> <p><b>Note:</b> Partial sensitivity indicates that the query in question has subqueries, and that some of the queries that make up the statement were implemented as insensitive while other queries were sensitive. This could happen, for example, if the outer query is insensitive but one or more of the subqueries were implemented as sensitive. All the query rules that apply to sensitive cursors apply also to partially sensitive cursors.</p>
SQLDFCODE	This field represents the type of SQL statement. Possible function code values are defined in <i>ISO SQL</i> as SQL statement codes.

Field Name	Usage
SQLDKKEYTYPE	<p>The type of key included in the select list for this statement. Possible values are:</p> <ul style="list-style-type: none"> <li>0 Key type when the descriptor is not describing the columns of a query; for example, a describe input. This value is set also if none of the columns of the query are members of a key.</li> <li>1 The select list includes all the columns of the primary key of the base table referenced by the query.</li> <li>2 The table referenced by the query does not have a primary key but the select list includes a set of columns that are defined as the preferred candidate key.</li> </ul> <p><b>Note:</b> The following, from a glossary of <i>ISO SQL</i> terms, defines <i>candidate keys</i>, with a reference to <i>preferred candidate key</i>: A key which may be used to identify a single row. There may be several candidate keys defined for a Base table, one of which may be the preferred candidate key. The preferred candidate key may be explicitly designated as the <i>primary key</i>. Thus, a candidate key is definable by any UNIQUE or PRIMARY KEY constraint. A candidate key which is not a primary key is an <i>alternate key</i>.</p>
SQLDRDBNAM	<p>This field contains the default RDB name to which this SQL statement belongs. This is the high-level qualifier for this SQL statement. The RDB name uniquely identifies the catalog associated with the statement. If the SQLDRDBNAM has a zero length, the RDB name is the RDBNAM used to access the RDB.</p>
SQLDSHEMA	<p>This field contains the default schema name to which this SQL statement belongs. This is the second-level qualifier for this statement.</p>

## 5.6.4.12 SQLDOPTGRP: SQL Descriptor Optional Group Description

**SQLDOPTGRP Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'02'	X'02'	X'D2'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
30	N-GDA X'76'	X'D2'

**SQLDOPTGRP Data**

			1 byte	2 bytes
<i>Descriptor Label</i>	<i>DRDA Type</i>	<i>Ref Type</i>	<i>Env LID</i>	<i>Length Override</i>
SQLUNNAMED	I2	X'02'	X'04'	2
SQLNAME_m	VCM	X'02'	X'3E'	255
SQLNAME_s	VCS	X'02'	X'32'	255
SQLLABEL_m	VCM	X'02'	X'3E'	255
SQLLABEL_s	VCS	X'02'	X'32'	255
SQLCOMMENTS_m	VCM	X'02'	X'3E'	255
SQLCOMMENTS_s	VCS	X'02'	X'32'	255
SQLUDTGRP	N-GDA	X'02'	X'5B'	0
SQLDXGRP	N-GDA	X'02'	X'D4'	0

Descriptor in HEX:077800050202D2 1E76D2 040002 3E00FF 3200FF 3E00FF  
3200FF 3E00FF 3200FF 5B0000 D40000

**Figure 5-39** SQLDOPTGRP Group Descriptor

This group defines pairs of variable-length character strings. SQLNAME\_m and SQLNAME\_s are mutually-exclusive; SQLLABEL\_m and SQLLABEL\_s are mutually-exclusive; SQLCOMMENTS\_m and SQLCOMMENTS\_s are mutually-exclusive; that is, only one non-zero value can be specified for the duplicated field. If both mutually-exclusive fields have a non-zero length, return an FD:OCA data mismatch reply message. (Refer to the DTAMCHRM term in the DDM Reference.)

Field Name	Usage
SQLNAME	Name of the column or variable as it appeared in the original SQL statement. This string can have any syntax that the requester can handle. The 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.

Field Name	Usage
SQLLABEL	Descriptive label associated with this column or variable. 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, return DTAMCHRM. 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, return DTAMCHRM.
SQLCOMMENTS	Comments are remarks (long description) associated with this column or variable.
SQLUNNAMED	This field can have a value of 0 or 1. This field is returned if the variable describes a column. A value of 1 indicates the column name is generated by the RDB. Otherwise, the value is 0. The unnamed attribute refers to the name of the column or variable and whether it is the derived name of the column from the select list, or whether the name is generated by the RDB.
SQLUDTGRP	This group is returned as a non-null group only if the column or variable is a user-defined type.
SQLDXGRP	This group is returned as a non-null group only if extended describe is requested as specified by the TYPSQLDA parameter on the command.

## 5.6.4.13 SQLDXGRP: SQL Descriptor Extended Group Description

**SQLDXGRP Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'02'	X'02'	X'D4'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
42	N-GDA X'76'	X'D4'

**SQLDXGRP Data**

			1 byte	2 bytes
<i>Descriptor Label</i>	<i>DRDA Type</i>	<i>Ref Type</i>	<i>Env LID</i>	<i>Length Override</i>
SQLXKEYMEM	I2	X'02'	X'04'	2
SQLXUPDATEABLE	I2	X'02'	X'04'	2
SQLXGENERATED	I2	X'02'	X'04'	2
SQLXPARMMODE	I2	X'02'	X'04'	2
SQLXRDBNAM	VCS	X'02'	X'32'	255
SQLXCORNAME_m	VCM	X'02'	X'3E'	255
SQLXCORNAME_s	VCS	X'02'	X'32'	255
SQLXBASENAME_m	VCM	X'02'	X'3E'	255
SQLXBASENAME_s	VCS	X'02'	X'32'	255
SQLXSHEMA_m	VCM	X'02'	X'3E'	255
SQLXSHEMA_s	VCS	X'02'	X'32'	255
SQLXNAME_m	VCM	X'02'	X'3E'	255
SQLXNAME_s	VCS	X'02'	X'32'	255

```
Descriptor in HEX:077800050202D4 2A76D4 040002 040002 040002 040002
                    3200FF 3200FF 3200FF 3200FF 3200FF
                    3E00FF 3200FF 3E00FF 3200FF
```

**Figure 5-40** SQLDXGRP Group Descriptor

This group defines pairs of variable-length character strings. The \*\_m and \*\_s strings are mutually-exclusive; that is, only one non-zero value can be specified for the duplicated field. If both mutually-exclusive fields have a non-zero length, return an FD:OCA data mismatch reply message. (Refer to the DTAMCHRM term in the DDM Reference.) If the SQLXRDBNAM is null, the value for the field is defaulted based on the value in the SQLDRDBNAM field in the SQL Descriptor Header Group (SQLDHGRP).

Field Name	Usage
SQLXKEYMEM	This field can have a value of 0 or 1. A value of 1 indicates that this is a column that is a member of the primary key or is the member of a unique index. Otherwise, the value is 0.
SQLXUPDATEABLE	This field can have a value of 0 or 1. A value of 1 indicates that this is a column that is updatable. Otherwise, the value is 0.

Field Name	Usage
SQLXGENERATED	<p>This field can have a value of 0 or 1, 2, 3, 4, or 5.</p> <ol style="list-style-type: none"> <li>1 Indicates that the data for this column was generated because of an expression.</li> <li>2 Indicates that the data for this column was generated because of an identity without a default value.</li> <li>3 Indicates that the data for the column was generated when the row was inserted into the table as a ROWID value without a default value.</li> <li>4 Indicates that the data for this column was generated because of an identity with a default value.</li> <li>5 Indicates the data for the column was generated when the row was inserted into the table as a ROWID value with a default value.</li> </ol> <p>Otherwise the value is 0.</p>
SQLXPARMMODE	<p>This field can have a value of 1, 2, or 4 if the field represents a parameter for a CALL statement:</p> <ol style="list-style-type: none"> <li>0 The field is not for use with a CALL statement.</li> <li>1 The field describes an input-only parameter.</li> <li>2 The field describes an input and output parameter.</li> <li>4 The field describes an output-only parameter.</li> </ol>
SQLXCORNAME	<p>This field contains the table correlation name related to the column. This field is only specified when there is a correlation name specified in the SQL statement; otherwise, the length of this field is zero.</p>
SQLXBASENAME	<p>This field contains the name of the underlying table or view referenced by the SQL statement if this column is related to a result set. If this field is a parameter of a CALL statement, then this field contains the procedure name. If the column is an expression or the result of a join, the length of this field is zero.</p>
SQLXRDBNAM	<p>This field contains the default RDB name to which this field belongs. This is the high-level qualifier for this field. The RDB name uniquely identifies the catalog associated with the statement. If the SQLXRDBNAM has a zero length, the SQLDRDBNAM identifies the value for this field if the SQLDHGRP is not null. If the SQLDRDBNAM has a zero length or the SQLDHGRP is null, the RDB name is the RDBNAM used to access the RDB.</p>
SQLXSHEMA	<p>This field contains the schema name to which this data object belongs. The SQLXSHEMA_m or SQLXSHEMA_s are mutually-exclusive. If the mixed and SBCS character strings have a zero length, the SQLDSHEMA identifies the default value for this field if the SQLDHGRP is not null.</p>
SQLXNAME	<p>This field contains the column or parameter name. This is the unqualified name for this field.</p>

## 5.6.4.14 SQLNUMEXT: SQL Extent Description for Variable Arrays

**SQLNUMEXT Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'03'	X'02'	X'76'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
6	X'71'	X'76'

**SQLNUMEXT Data**

		1 byte	1 byte	1 byte
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Row LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLEXTROW	X'01'	X'66'	0 (all)	0 (all)

Descriptor in HEX:07780005030276 067176 660000

**Figure 5-41** SQLNUMEXT Row Descriptor

5.6.4.15 SQLEXTROW: SQL Array Row Description for a Variable Array

**SQLEXTROW Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'03'	X'02'	X'66'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
6	X'71'	X'66'

**SQLEXTROW Data**

		1 byte	1 byte	1 byte
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Group LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLEXTGRP	X'01'	X'71'	0 (all)	1

Descriptor in HEX:07780005030266 067166 0001

**Figure 5-42** SQLEXTROW Row Descriptor

## 5.6.4.16 SQLEXTGRP: SQL Extent Group Description for a Variable Array

**SQLEXTGRP Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'03'	X'02'	X'56'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
9	X'75'	X'56'

**SQLEXTGRP Data**

			1 byte	2 bytes
<i>Descriptor Label</i>	<i>DRDA Type</i>	<i>Ref Type</i>	<i>Env LID</i>	<i>Length Override</i>
ARRAY_EXT	14	X'02'	X'02'	4

Descriptor in HEX:07780005030256 097556 020004 020004

**Figure 5-43** SQLEXTGRP Row Descriptor

Field Name	Usage
ARRAY_EXT	The extent specification for the variable. The extent provides the number of data elements that are required to be provided for the variable. It is the dimension of the array of the type described. Each SDA described in the FDODSC in relative order is required to contain an extent value.
ARRAY_OFF	The offset value is an index to the start of each input variable array. It is the relative byte count from the start of the FDODTA data object to the first byte of the first element of the variable array. The offset value is from the beginning of the FDODTA length field (LL). Each SDA described in the FDODSC in relative order is required to contain an offset value.

5.6.4.17 SQLDIAGSTT: SQL Diagnostics Statement Group Description

**SQLDIAGSTT Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'02'	X'02'	X'D1'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
51	N-GDA X'76'	X'D3'

**SQLDIAGSTT Data**

			1 byte	2 bytes
<i>Descriptor Label</i>	<i>DRDA Type</i>	<i>Ref Type</i>	<i>Env LID</i>	<i>Length Override</i>
SQLDSFCOD	I4	X'02'	X'02'	4
SQLDSCOST	I4	X'02'	X'02'	4
SQLDSLROW	I4	X'02'	X'02'	4
SQLDSNPM	I4	X'02'	X'02'	4
SQLDSNRS	I4	X'02'	X'02'	4
SQLDSRNS	I4	X'02'	X'02'	4
SQLSDCOD	I4	X'02'	X'02'	4
SQLDSROWC	I8	X'02'	X'16'	8
SQLDSNROW	I8	X'02'	X'16'	8
SQLDSROWCS	I8	X'02'	X'16'	8
SQLDSACON	FCS	X'02'	X'30'	1
SQLDSACRH	FCS	X'02'	X'30'	1
SQLDSACRS	FCS	X'02'	X'30'	1
SQLDSACSL	FCS	X'02'	X'30'	1
SQLDSACSE	FCS	X'02'	X'30'	1
SQLDSACTY	FCS	X'02'	X'30'	1
SQLDSCERR	FCS	X'02'	X'30'	1
SQLDSMORE	FCS	X'02'	X'30'	1

```
Descriptor in HEX:077800050202D3 3376D3 020004 020004 020004 020004
                                020004 020004 020004 160008 160008
                                160008 300001 300001 300001 300001
                                300001 300001 300001 300001
```

**Figure 5-44** SQLDIAGSTT Group Descriptor

Table 5-7 SQLDIAGSTT Field Descriptions

Field Name	Usage
SQLDSCOST	COST_ESTIMATE: For a PREPARE statement, contains a relative number estimate of the resources required for every execution. It does not reflect an estimate of the time required. When preparing a dynamically defined statement, this value can be used as an indicator of the relative cost of the prepared statement. The value varies depending on changes to statistics in the catalog and can vary between releases of the product. It is an estimated cost for the access plan chosen by the optimizer. The value is zero if not a PREPARE statement or if no estimate is available.
SQLSDCOD	DYNAMIC_FUNCTION_CODE: If the previous SQL statement was an EXECUTE, returns an integer that identifies the prepared statement that was executed. Possible values are identified in <a href="#">Appendix D</a> (on page 721). If not known, the value zero is returned.
SQLSFCOD	FUNCTION_CODE: Returns an integer that identifies the previous SQL statement. Possible values are identified in <a href="#">Appendix D</a> (on page 721). If not known, the value zero is returned.
SQLDSLROW	LAST_ROW: For a multiple-row FETCH statement, contains a value of SQLSTATE 02000 in the SQLCARD if the last row currently in the table is in the set of rows that have been fetched. For cursors that are not sensitive to updates, there would be no need to do a subsequent FETCH, since the result would be an end of data indication (SQLSTATE 02000). For cursors that are sensitive to updates, a subsequent FETCH may return more data if the row had been inserted before the FETCH was executed. This option is not applicable for single-row FETCH statements because if it is the last row that is FETCHED, the FETCH is like any other that returned data. In a multi-row FETCH case, when the full set of rows requested was not returned because the set included the last row in the table, this option indicates that this is what happened and it gives the application the opportunity to not bother issuing the next FETCH that, for the INSENSITIVE CURSOR case, will return SQLSTATE 02000. Otherwise, the value zero is returned.
SQLDSNPM	NUMBER_PARAMETER_MARKER: For a PREPARE statement, contains the number of parameter markers in the prepared statement. Otherwise, the value zero is returned.
SQLDSNROW	NUMBER_ROWS: If the previous SQL statement was an OPEN or a FETCH which caused the size of the result table to be known, this field identifies the number of rows in the result table. Otherwise, the value zero is returned.
SQLDSNRS	NUMBER_RESULT_SETS: For a CALL statement, contains the actual number of result sets returned by the procedure. Otherwise, the value zero is returned.

Field Name	Usage
SQLDSRNS	<p>RETURN_STATUS: Identifies the status value returned from the stored procedure associated with the previously executed SQL statement, provided that the statement was a CALL statement invoking a procedure that returns a status. If the previous statement is not such a statement, the value returned has no meaning and could be any integer.</p>
SQLDSROWC	<p>ROW_COUNT: Identifies the number of rows associated with the previous SQL statement that was executed.</p> <ol style="list-style-type: none"> <li>1. If the previous statement is a DELETE, INSERT, or UPDATE statement, this field identifies the number of rows deleted, inserted, or updated by that statement, excluding rows affected by either triggers or referential integrity constraints.</li> <li>2. If the previous statement is a multiple-row FETCH, this field identifies the number of rows fetched.</li> <li>3. If the previous statement is a PREPARE statement, this field identifies the estimated number of result rows in the prepared statement.</li> </ol> <p>Otherwise, the value zero is returned.</p>
SQLDSROWCS	<p>ROW_COUNT_SECONDARY: Identifies the number of rows associated with secondary actions from the previous SQL statement that was executed. Otherwise, the value zero is returned.</p>
SQLDSACON	<p>SQL_ATTR_CONCURRENCY: For an ALLOCATE or OPEN statement, indicates the concurrency control option of read-only, locking, optimistic using timestamps, or optimistic using values.</p> <ul style="list-style-type: none"> <li>• R indicates read-only.</li> <li>• L indicates locking.</li> <li>• T indicates comparing row versions using timestamps or rowids.</li> <li>• V indicates comparing values.</li> <li>• Blank otherwise.</li> </ul>
SQLDSACRH	<p>SQL_ATTR_CURSOR_HOLD: For an ALLOCATE or OPEN statement, indicates cursor holdability, whether a cursor can be held open across multiple units of work or not.</p> <ul style="list-style-type: none"> <li>• N indicates that this cursor will not remain open across multiple units of work.</li> <li>• Y indicates that this cursor will remain open across multiple units of work.</li> <li>• Blank otherwise.</li> </ul>

Field Name	Usage
SQLDSACRS	<p>SQL_ATTR_CURSOR_ROWSET: For an ALLOCATE or OPEN statement, indicates rowset accessibility, whether a cursor can be accessed using rowset positioning or not.</p> <ul style="list-style-type: none"> <li>• N indicates that this cursor only supports row positioned operations.</li> <li>• Y indicates that this cursor supports rowset positioned operations.</li> <li>• Blank otherwise.</li> </ul>
SQLDSACSE	<p>SQL_ATTR_CURSOR_SENSITIVITY: For an ALLOCATE or OPEN statement, indicates cursor sensitivity, whether a cursor does or does not show updates to cursor rows made by other connections.</p> <p>A Indicates asensitive.  I Indicates insensitive.  S Indicates sensitive.  P Indicates partially sensitive.</p> <p>Note that all the query rules that apply to sensitive cursors apply also to partially sensitive cursors.</p> <p>Otherwise, blank.</p>
SQLDSACSL	<p>SQL_ATTR_CURSOR_SCROLLABLE: For an ALLOCATE or OPEN statement, indicates cursor scrollability, whether a cursor can be scrolled forward and backward or not.</p> <ul style="list-style-type: none"> <li>• N indicates that this cursor is not scrollable.</li> <li>• Y indicates that this cursor is scrollable.</li> <li>• Blank otherwise.</li> </ul>
SQLDSACTY	<p>SQL_ATTR_CURSOR_TYPE: For an ALLOCATE or OPEN statement, indicates the type of cursor, whether a cursor type is dynamic or static.</p> <ul style="list-style-type: none"> <li>• D indicates a dynamic cursor.</li> <li>• F indicates a forward-only cursor.</li> <li>• S indicates a static cursor.</li> <li>• Blank otherwise.</li> </ul>
SQLDSCERR	<p>CONVERSION_ERROR:</p> <ul style="list-style-type: none"> <li>• 1 indicates there was a conversion error when converting a character data value for one of the diagnostic fields.</li> <li>• Blank otherwise.</li> </ul>
SQLDSMORE	<p>MORE:</p> <ul style="list-style-type: none"> <li>• N indicates that all the warnings and errors from the previous SQL statement were stored in the diagnostic area.</li> <li>• Y indicates that some of the warnings and errors from the previous SQL statement were discarded.</li> </ul>

5.6.4.18 SQLCNGRP: SQL Diagnostics Connection Group Description

**SQLCNGRP Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'02'	X'02'	X'D6'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
27	GDA X'75'	X'D6'

**SQLCNGRP Data**

			1 byte	2 bytes
<i>Descriptor Label</i>	<i>DRDA Type</i>	<i>Ref Type</i>	<i>Env LID</i>	<i>Length Override</i>
SQLCNSTATE	I4	X'02'	X'02'	4
SQLCNSTATUS	I4	X'02'	X'02'	4
SQLCNATYPE	FCS	X'02'	X'30'	1
SQLCNETYPE	FCS	X'02'	X'30'	1
SQLCNPRDID	FCS	X'02'	X'30'	8
SQLCNRDB	VCS	X'02'	X'32'	255
SQLCNCLASS	VCS	X'02'	X'32'	255
SQLCNAUTHID	VCS	X'02'	X'32'	255

Descriptor in HEX:077800050202D6 1B75D6 020004 020004 300001 300001  
 300008 3200FF 3200FF 3200FF

**Figure 5-45 SQLCNGRP Group Descriptor**

**Table 5-8** SQLCNGRP Field Descriptions

Field Name	Usage
SQLCNSTATE	CONNECTION_STATE: Contains a value of -1 if the connection is unconnected; 1 if the connection is connected to an RDB. Otherwise, the value zero is returned.
SQLCNSTATUS	CONNECTION_STATUS: Contains a value of 1 if committable updates can be performed on the connection for this unit of work; 2 if no committable updates can be performed on the connection for this unit of work. Otherwise, the value zero is returned.
SQLCNATYPE	AUTHENTICATION_TYPE: Contains a type value of 'S' for a server authentication; 'C' for client authentication; blank for unspecified authentication.
SQLCNETYPE	ENCRYPTION_TYPE: Contains a type value of 'P' for password encryption;
SQLCNPRDID	PRDID: Contains the registered server product signature. The form is <i>pppvrrm</i> , where: <ul style="list-style-type: none"> <li>• <i>ppp</i> identifies the product. This value should be registered with The Open Group.</li> <li>• <i>vv</i> is a two-digit version identifier such as '04'.</li> <li>• <i>rr</i> is a two-digit release identifier such as '01'.</li> <li>• <i>m</i> is a one-digit modification level such as '0'.</li> </ul>
SQLCNRDB	RDBNAM: Contains the RDB name of the server.
SQLCNCLASS	CLASS_NAME: Contains the registered RDB server class name. Refer to the DDM Reference for the EXCSATRD class name.
SQLCNAUTHID	AUTHID: Authorization identifier used by connected server. This may be different than the local user ID because of user ID translation and authorization exits.

5.6.4.19 SQLDCGRP: SQL Diagnostics Condition Group Description

**SQLDCGRP Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'05'	X'02'	X'D5'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
78	GDA X'75'	X'D5'

**SQLDCGRP Data**

			1 byte	2 bytes
<i>Descriptor Label</i>	<i>DRDA Type</i>	<i>Ref Type</i>	<i>Env LID</i>	<i>Length Override</i>
SQLDCCODE	I4	X'02'	X'02'	4
SQLDCSTATE	FCS	X'02'	X'30'	5
SQLDCREASON	I4	X'02'	X'02'	4
SQLDCLINEN	I4	X'02'	X'02'	4
SQLDCROWN	I8	X'02'	X'16'	8
SQLDCER01	I4	X'02'	X'02'	4
SQLDCER02	I4	X'02'	X'02'	4
SQLDCER03	I4	X'02'	X'02'	4
SQLDCER04	I4	X'02'	X'02'	4
SQLDCPART	I4	X'02'	X'02'	4
SQLDCPPPOP	I4	X'02'	X'02'	4
SQLDCMSGID	FCS	X'02'	X'30'	10
SQLDCMDE	FCS	X'02'	X'30'	8
SQLDCPMOD	FCS	X'02'	X'30'	5
SQLDCRDB	VCS	X'02'	X'32'	255
SQLDCTOKS	N-RLO	X'02'	X'F7'	0
SQLDCMSG_m	NVCM	X'02'	X'3F'	32,672
SQLDCMSG_s	NVCS	X'02'	X'33'	32,672
SQLDCCOLN_m	NVCM	X'02'	X'3F'	255
SQLDCCOLN_s	NVCS	X'02'	X'33'	255
SQLDCCURN_m	NVCM	X'02'	X'3F'	255
SQLDCCURN_s	NVCS	X'02'	X'33'	255
SQLDCPNAM_m	NVCM	X'02'	X X'3F'	255
SQLDCPNAM_s	NVCS	X'02'	X'33'	255
SQLDCXGRP	N-GDA	X'02'	X'D3'	1

```
Descriptor in HEX:077800050202D5 4E75D5 020004 300005 020004 020004  
160008 020004 020004 020004 020004 020004  
020004 020004 30000A 300008 300005  
F70000 3F7FA0 337FA0 3F00FF 3300FF  
3F00FF 3300FF 3F00FF 3300FF D30001
```

**Figure 5-46** SQLDCGRP Group Descriptor

Table 5-9 SQLDCGRP Field Descriptions

Field Name	Usage
SQLDCCODE	SQLCODE: Returns the SQLCODE for the specified diagnostic.
SQLDCSTATE	SQLSTATE: Returns the SQLSTATE for the specified diagnostic.
SQLDCREASON	REASON_CODE: Contains the reason code for errors that have a reason code token in the message text. Otherwise, the value zero is returned.
SQLDCLINEN	LINE_NUMBER: Contains the line number where the error was encountered in an SQL procedure where an error is encountered parsing the SQL procedure body. Otherwise, the value zero is returned.
SQLDCROWN	ROW_NUMBER: Returns the number of the row where the condition was encountered, when such a value is available and applicable. Otherwise, the value zero is returned.
SQLDCER01	ERROR_CODE1: Contains an internal error code. Otherwise, the value zero is returned.
SQLDCER02	ERROR_CODE2: Contains an internal error code. Otherwise, the value zero is returned.
SQLDCER03	ERROR_CODE3: Contains an internal error code. Otherwise, the value zero is returned.
SQLDCER04	ERROR_CODE4: Contains an internal error code. Otherwise, the value zero is returned.
SQLDCPART	PARTITION_NUMBER: For a partitioned database, contains the partition number of the partition that encountered the error or warning. Otherwise, the value zero is returned.
SQLDCPPPOP	PARAMETER_ORDINAL_NUMBER: Condition is related to the <i>i</i> th parameter of the CALL, the value of <i>i</i> is returned. Otherwise, zero is returned.
SQLDCMSGID	MESSAGE_ID: Server-specific message identifier that corresponds to the message text if provided. Otherwise, blanks are returned.
SQLDCMDE	MODULE_DETECTING_ERROR: Identifier indicating which module detected the error. Otherwise, blanks are returned.
SQLDCPMOD	PARAMETER_MODE: Related to the <i>i</i> th parameter of the CALL, the parameter mode (IIN, OUT, or INOUT) of the <i>i</i> th parameter is returned. Otherwise, blanks are returned.
SQLDCRDB	RDBNAM: RDB name of the server that generated the condition.
SQLDCTOKS	MESSAGE_TOKENS: Message token array.
SQLDCMSG	MESSAGE_TEXT: Condition-related message text. If message text is not null, SQLDMSGID must be provided to uniquely identify the message text. This field must be NULL, if DIAGLVL2 is specified on the ACCRDB command.
SQLDCCOLN	COLUMN_NAME: If condition was caused by an inaccessible column, the name of the column that caused the error is returned. Otherwise, the null string is returned.

Field Name	Usage
SQLDCCURN	CURSOR_NAME: If condition was caused by an invalid cursor, the name of the cursor is returned. Otherwise, the null string is returned.
SQLDCPNAM	PARAMETER_NAME: If condition is related to the <i>i</i> th parameter of the CALL, and a parameter name was specified for the parameter when the routine was created, the parameter name of the <i>i</i> th parameter is returned. Otherwise, the null string is returned.
SQLDCXGRP	EXTENDED_NAMES: Extended Diagnostic Name Group.

5.6.4.20 SQLDCXGRP: SQL Diagnostics Extended Names Group Description

**SQLDCXGRP Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'02'	X'02'	X'D8'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
69	N-GDA X'76'	X'D8'

**SQLDCXGRP Data**

			1 byte	2 bytes
<i>Descriptor Label</i>	<i>DRDA Type</i>	<i>Ref Type</i>	<i>Env LID</i>	<i>Length Override</i>
SQLDCXRDB	VCS	X'02'	X'32'	255
SQLDCXSCH_m	NVCM	X'02'	X'3F'	255
SQLDCXSCH_s	NVCS	X'02'	X'33'	255
SQLDCXNAM_m	NVCM	X'02'	X'3F'	255
SQLDCXNAM_s	NVCS	X'02'	X'33'	255
SQLDCXTBLN_m	NVCM	X'02'	X'3F'	255
SQLDCXTBLN_s	NVCS	X'02'	X'33'	255
SQLDCXCRDB	VCS	X'02'	X'32'	255
SQLDCXCSCHE_m	NVCM	X'02'	X'3F'	255
SQLDCXCSCHE_s	NVCS	X'02'	X'33'	255
SQLDCXCNAM_m	NVCM	X'02'	X'3F'	255
SQLDCXCNAM_s	NVCS	X'02'	X'33'	255
SQLDCXRRDB	VCS	X'02'	X'32'	255
SQLDCXRSCH_m	NVCM	X'02'	X'3F'	255
SQLDCXRSCH_s	NVCS	X'02'	X'33'	255
SQLDCXRNAM_m	NVCM	X'02'	X'3F'	255
SQLDCXRNAM_s	NVCS	X'02'	X'33'	255
SQLDCXTRDB	VCS	X'02'	X'32'	255
SQLDCXTSCH_m	NVCM	X'02'	X'3F'	255
SQLDCXTSCH_s	NVCS	X'02'	X'33'	255
SQLDCXTNAM_m	NVCM	X'02'	X'3F'	255
SQLDCXTNAM_s	NVCS	X'02'	X'33'	255

```
Descriptor in HEX:077800050202D8 4576D8 3200FF 3F00FF 3300FF 3F00FF
3300FF 3F00FF 3300FF 3200FF 3F00FF
3300FF 3F00FF 3300FF 3200FF 3F00FF
3300FF 3F00FF 3300FF 3200FF 3F00FF
3300FF 3F00FF 3300FF
```

**Figure 5-47 SQLDCXGRP Group Descriptor**

Table 5-10 SQLDCXGRP Field Descriptions

Field Name	Usage
SQLDCXRDB	OBJECT_RDBNAM: Contains the RDB name that contains the object that caused the condition. Otherwise, a zero-length string is returned.
SQLDCXSCH	OBJECT_SCHEMA: Contains the schema name that contains the object that caused the condition. Otherwise, the empty string is returned.
SQLDCXNAME	SPECIFIC_NAME: Contains the specific name of the object that caused the condition. The specific name is qualified by SQLDCXRDB and SQLDCXSCH fields. Otherwise, the empty string is returned.
SQLDCXTBLN	TABLE_NAME: Contains the table name of the object that caused the condition. The table name is uniquely identified by the SQLDCXRDB and SQLDCXSCH fields. Otherwise, the empty string is returned.
SQLDCXCRDB	CONSTRAINT_RDBNAM: Contains the RDB name that contains the constraint that caused the condition. Otherwise, the empty string is returned.
SQLDCXCSCHE	CONSTRAINT_SCHEMA: Contains the schema name that contains the object that caused the condition. Otherwise, the empty string is returned.
SQLDCXCNAM	CONSTRAINT_NAME: Contains the constraint name of the object that caused the condition. The constraint name is uniquely identified by the SQLDCXCRDB and SQLDCXCSCHE fields. Otherwise, the empty string is returned.
SQLDCXRRDB	ROUTINE_RDBNAM: Contains the RDB name that contains the routine that caused the condition. Otherwise, a zero-length string is returned.
SQLDCXRSCH	ROUTINE_SCHEMA: Contains the schema name that contains the routine that caused the condition. Otherwise, the empty string is returned.
SQLDCXRNAM	ROUTINE_NAME: Contains the routine name of the object that caused the condition. The constraint name is uniquely identified by the SQLDCXRRDB and SQLDCXRSCH fields. Otherwise, the empty string is returned.
SQLDCXTRDB	TRIGGER_RDBNAM: Contains the RDB name that contains the trigger that caused the condition. Otherwise, a zero-length string is returned.
SQLDCXTSCH	TRIGGER_SCHEMA: Contains the schema name that contains the trigger that caused the condition. Otherwise, the empty string is returned.
SQLDCXTNAM	TRIGGER_NAME: Contains the trigger name of the object that caused the condition. The constraint name is uniquely identified by the SQLDCXTRDB and SQLDCXTSCH fields. Otherwise, the empty string is returned.

5.6.4.21 SQLTOKGRP: SQL Diagnostics Token Group

**SQLTOKGRP Meta Data and Data Descriptor**

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
<i>MDD Length</i>	<i>MDD Type</i>	<i>Identity</i>	<i>Class</i>	<i>MD Type</i>	<i>MD Ref Type</i>	<i>DRDA Type</i>
7	X'78'	0	X'05'	X'03'	X'02'	X'D7'

byte 7	byte 8	byte 9
<i>Data Length</i>	<i>Data Type</i>	<i>Identity</i>
9	X'71'	X'D7'

**SQLTOKGRP Data**

		byte 10	byte 11	byte 12
<i>Descriptor Name</i>	<i>Ref Type</i>	<i>Group LID</i>	<i>Elem_Taken</i>	<i>Rep_Factor</i>
SQLDCTOK_m	NVCM	X'02'	X'3F'	255
SQLDCTOK_s	NVCS	X'02'	X'33'	255

Descriptor in HEX:077800050302D7 0971D7 3F00FF 3300FF

**Figure 5-48** SQLTOKGRP Group Descriptor

### 5.6.5 Early Environmental Descriptors

Figure 5-49 through Figure 5-83 show how each of the DRDA database management system environments represent each of the DRDA and SQL data types.

Figure 5-49 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'02'	X'32' (VCS) X'33' (NVCS)
---	--------------	---	-----------------	--------------------	--------------------	-----------------------------

FD	FD:OCA	FD:OCA	FD:OCA	CCSID	Chr Mode	Fld	Length
Tr	Tripl	Tripl	Field		Siz		
Ln	'SDA'	LID	Type				
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 5-49** 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 5-49 (on page 333).

**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<sup>46</sup> (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'.

---

46. An example of an QTDSQLASC machine is the IBM RS/6000, where basic floating point numbers are in IEEE floating point format and numbers are not byte-reversed. This contrasts with a QTDSQLX86 machine where basic floating numbers are in byte-reversed IEEE floating point format and integers are also byte-reversed.

**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 CCSIDDBC, CCSIDMBC, or CCSIDXML) 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. TYPDEFOVR 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 385).)

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 32,767 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 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'02'	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	010203						
(QTDSQL370 nullable form)				0C7003A3	00000000	00000004					

**Figure 5-50** 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

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'02'	X'04' (I2) X'05' (NI2)
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'04'	X'23'	0	0	0 2
12	X'70'	X'05'	X'A3'	0	0	0 2
QTDSQL400 (AS/400 Processors)						
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)						
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)						
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)						
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 010205						
(QTDSQL370 nullable form) 0C7005A3 00000000 00000002						

Figure 5-51 DRDA Type X'04,05' SQL Type 500,501 SMALL INTEGER

5.6.5.3 One-Byte Integer

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'02'	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	010207						
(QTDSQL370 nullable form)				0C7007A3	00000000	00000001					

Figure 5-52 DRDA Type X'06,07' SQL Type n/a,n/a

5.6.5.4 Sixteen-Byte Basic Float

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'02'	X'08' (BF16) X'09' (NBF16)					
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'08'	X'40'	0	0	0	0	0	0	16	
12	X'70'	X'09'	X'C0'	0	0	0	0	0	0	16	
QTDSQL400 (AS/400 Processors)											
12	X'70'	X'08'	X'48'	0	0	0	0	0	0	16	
12	X'70'	X'09'	X'C8'	0	0	0	0	0	0	16	
QTDSQLX86 (Intel 80X86 Processors)											
12	X'70'	X'08'	X'47'	0	0	0	0	0	0	16	
12	X'70'	X'09'	X'C7'	0	0	0	0	0	0	16	
QTDSQLASC (IEEE ASCII Processors)											
12	X'70'	X'08'	X'48'	0	0	0	0	0	0	16	
12	X'70'	X'09'	X'C8'	0	0	0	0	0	0	16	
QTDSQLVAX (VAX Processors)											
12	X'70'	X'08'	X'49'	0	0	0	0	0	0	16	
12	X'70'	X'09'	X'C9'	0	0	0	0	0	0	16	
Example Descriptor in Hex    07780005 010209											
(QTDSQL370 nullable form)    0C7009C0 00000000 00000010											

Figure 5-53 DRDA Type X'08,09' SQL Type 480,481 FLOAT (16)

5.6.5.5 Eight-Byte Basic Float

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'02'	X'0A' (BF8) X'0B' (NBF8)			
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'0A'	X'40'	0	0	0	0	8	
12	X'70'	X'0B'	X'C0'	0	0	0	0	8	
QTDSQL400 (AS/400 Processors)									
12	X'70'	X'0A'	X'48'	0	0	0	0	8	
12	X'70'	X'0B'	X'C8'	0	0	0	0	8	
QTDSQLX86 (Intel 80X86 Processors)									
12	X'70'	X'0A'	X'47'	0	0	0	0	8	
12	X'70'	X'0B'	X'C7'	0	0	0	0	8	
QTDSQLASC (IEEE ASCII Processors)									
12	X'70'	X'0A'	X'48'	0	0	0	0	8	
12	X'70'	X'0B'	X'C8'	0	0	0	0	8	
QTDSQLVAX (VAX Processors)									
12	X'70'	X'0A'	X'49'	0	0	0	0	8	
12	X'70'	X'0B'	X'C9'	0	0	0	0	8	
Example Descriptor in Hex    07780005 01020B (QTDSQL370 nullable form)    0C700BC0 00000000 00000008									

Figure 5-54 DRDA Type X'0A,0B' SQL Type 480,481 FLOAT (8)

5.6.5.6 Four-Byte Basic Float

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'02'	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	0	0	0	4	
12	X'70'	X'0D'	X'C0'	0	0	0	0	0	0	4	
QTDSQL400 (AS/400 Processors)											
12	X'70'	X'0C'	X'48'	0	0	0	0	0	0	4	
12	X'70'	X'0D'	X'C8'	0	0	0	0	0	0	4	
QTDSQLX86 (Intel 80X86 Processors)											
12	X'70'	X'0C'	X'47'	0	0	0	0	0	0	4	
12	X'70'	X'0D'	X'C7'	0	0	0	0	0	0	4	
QTDSQLASC (IEEE ASCII Processors)											
12	X'70'	X'0C'	X'48'	0	0	0	0	0	0	4	
12	X'70'	X'0D'	X'C8'	0	0	0	0	0	0	4	
QTDSQLVAX (VAX Processors)											
12	X'70'	X'0C'	X'49'	0	0	0	0	0	0	4	
12	X'70'	X'0D'	X'C9'	0	0	0	0	0	0	4	
Example Descriptor in Hex    07780005 01020D											
(QTDSQL370 nullable form)    0C700DC0 00000000 00000004											

Figure 5-55 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'02'	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	X'70'	X'0E'	X'30'	0	0	0	31	31
12	X'70'	X'0F'	X'B0'	0	0	0	31	31
QTDSQL400 (AS/400 Processors)								
12	X'70'	X'0E'	X'30'	0	0	0	31	31
12	X'70'	X'0F'	X'B0'	0	0	0	31	31
QTDSQLX86 (Intel 80X86 Processors)								
12	X'70'	X'0E'	X'30'	0	0	0	31	31
12	X'70'	X'0F'	X'B0'	0	0	0	31	31
QTDSQLASC (IEEE ASCII Processors)								
12	X'70'	X'0E'	X'30'	0	0	0	31	31
12	X'70'	X'0F'	X'B0'	0	0	0	31	31
QTDSQLVAX (VAX Processors)								
12	X'70'	X'0E'	X'30'	0	0	0	31	31
12	X'70'	X'0F'	X'B0'	0	0	0	31	31
Example Descriptor in Hex								
(QTDSQL370 nullable form)				07780005	01020F			
				0C700FB0	00000000	00001F1F		

Figure 5-56 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'02'	X'10' (ZD) X'11' (NZD)		
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	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								
(QTDSQL370 nullable form)				07780005	010211	0C7011B3	00000000	00001F1F

Figure 5-57 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)								
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'02'	X'12' (N) X'13' (NN)		
FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Mode	Fld Length and Prec;/Scale		
0	1	2	3	4 5 6 7	8 9	10 11		
QTDSQL370 (System/370 Processors)								
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)								
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)								
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)								
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)								
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 010213								
(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 5-58 DRDA Type X'12,13' SQL Type 504,505 NUMERIC CHARACTER

5.6.5.10 Result Set Locator

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'02'	X'14' (RSL) X'15' (NRSL)					
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'14'	X'23'	0	0	0	4				
12	X'70'	X'15'	X'A3'	0	0	0	4				
QTDSQL400 (AS/400 Processors)											
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)											
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)											
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)											
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 010215											
(QTDSQL370 nullable form)    0C7015A3 00000000 00000004											

Figure 5-59 DRDA Type X'14,15' SQL Type 972,973 RESULT SET LOCATOR

5.6.5.11 Eight-Byte Integer

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'02'	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	X'70'	X'16'	X'23'	0	0	0 0 8
12	X'70'	X'17'	X'A3'	0	0	0 0 8
QTDSQL400 (AS/400 Processors)						
12	X'70'	X'16'	X'23'	0	0	0 0 8
12	X'70'	X'17'	X'A3'	0	0	0 0 8
QTDSQLX86 (Intel 80X86 Processors)						
12	X'70'	X'16'	X'24'	0	0	0 0 8
12	X'70'	X'17'	X'A4'	0	0	0 0 8
QTDSQLASC (IEEE ASCII Processors)						
12	X'70'	X'16'	X'23'	0	0	0 0 8
12	X'70'	X'17'	X'A3'	0	0	0 0 8
QTDSQLVAX (VAX Processors)						
12	X'70'	X'16'	X'24'	0	0	0 0 8
12	X'70'	X'17'	X'A4'	0	0	0 0 8
Example Descriptor in Hex 07780005 010217						
(QTDSQL370 nullable form) 0C7017A3 00000000 00000008						

Figure 5-60 DRDA Type X'16,17' SQL Type 492,493 EIGHT-BYTE INTEGER

5.6.5.12 Large Object Bytes Locator

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'02'	X'18' (OBL) X'19' (NOBL)

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'18'	X'01'	0	0	0	4
12	X'70'	X'19'	X'81'	0	0	0	4

QTDSQL400 (AS/400 Processors)

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)

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)

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)

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 010219  
 (QTDSQL370 nullable form)    0C701981 00000000 00000004

**Figure 5-61** DRDA Type X'18,19' SQL Type 960,961 LARGE OBJECT BYTES LOCATOR

5.6.5.13 Large Object Character Locator

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'02'	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 01021B  
 (QTDSQL370 nullable form)    0C701B81 00000000 00000004

**Figure 5-62** DRDA Type X'1A,1B' SQL Type 964,965 LARGE OBJ. CHAR. SBCS LOCATOR

5.6.5.14 Large Object Character DBCS Locator

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'02'	X'1C' (OCDL) X'1D' (NOCDL)

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'1C'	X'01'	0	0	0	4
12	X'70'	X'1D'	X'81'	0	0	0	4

QTDSQL400 (AS/400 Processors)

12	X'70'	X'1C'	X'01'	0	0	0	4
12	X'70'	X'1D'	X'81'	0	0	0	4

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'1C'	X'01'	0	0	0	4
12	X'70'	X'1D'	X'81'	0	0	0	4

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'1C'	X'01'	0	0	0	4
12	X'70'	X'1D'	X'81'	0	0	0	4

QTDSQLVAX (VAX Processors)

12	X'70'	X'1C'	X'01'	0	0	0	4
12	X'70'	X'1D'	X'81'	0	0	0	4

Example Descriptor in Hex    07780005 01021D  
 (QTDSQL370 nullable form)  0C701D81 00000000 00000004

**Figure 5-63** DRDA Type X'1C,1D' SQL Type 968,969 LARGE OBJ. CHAR. DBCS LOCATOR

5.6.5.15 Row Identifier

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'02'	X'1E' (RI) X'1F' (NRI)					
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'1E'	X'02'	0	0	1	40				
12	X'70'	X'1F'	X'82'	0	0	1	40				
QTDSQL400 (AS/400 Processors)											
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)											
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)											
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)											
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	01021F						
(QTDSQL370 nullable form)				0C701F82	00000000	00010028					

Figure 5-64 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)						
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'02'	X'20' (D) X'21' (ND)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Res-Siz	erved	Fld	Length
0	1	2	3	4 5 6 7	8	9	10	11

QTDSQL370 (System/370 Processors)

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)

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)

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)

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)

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 010221  
 (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 5-65 DRDA Type X'20,21' SQL Type 384,385 DATE

5.6.5.17 Time

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'02'	X'22' (T) X'23' (NT)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Res-Siz	erved	Fld	Length
0	1	2	3	4 5 6 7	8	9	10	11

QTDSQL370 (System/370 Processors)

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)

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)

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)

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)

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 010223  
 (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 5-66 DRDA Type X'22,23' SQL Type 388,389 TIME

5.6.5.18 Timestamp

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'02'	X'24' (TS) X'25' (NTS)	
FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Res-Siz	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 010225							
(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 5-67 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'02'	X'26' (FB) X'27' (NFB)					
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'26'	X'01'	0	0	0	32767				
12	X'70'	X'27'	X'81'	0	0	0	32767				
QTDSQL400 (AS/400 Processors)											
12	X'70'	X'26'	X'01'	0	0	0	32767				
12	X'70'	X'27'	X'81'	0	0	0	32767				
QTDSQLX86 (Intel 80X86 Processors)											
12	X'70'	X'26'	X'01'	0	0	0	32767				
12	X'70'	X'27'	X'81'	0	0	0	32767				
QTDSQLASC (IEEE ASCII Processors)											
12	X'70'	X'26'	X'01'	0	0	0	32767				
12	X'70'	X'27'	X'81'	0	0	0	32767				
QTDSQLVAX (VAX Processors)											
12	X'70'	X'26'	X'01'	0	0	0	32767				
12	X'70'	X'27'	X'81'	0	0	0	32767				
Example Descriptor in Hex 07780005 010227											
(QTDSQL370 nullable form) 0C702781 00000000 00007FFF											

Figure 5-68 DRDA Type X'26,27' SQL Type 452,453 FIXED BYTES

5.6.5.20 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'02'	X'28' (VB) X'29' (NVB)	
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	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)							
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)							
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)							
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)							
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							
(QTDSQL370 nullable form)				07780005	010229		
				0C702982	00000000	00017FFF	

Figure 5-69 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'02'	X'2A' (LVB) X'2B' (NLVB)	
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	X'70'	X'2A'	X'02'	0	0	1	32767
12	X'70'	X'2B'	X'82'	0	0	1	32767
QTDSQL400 (AS/400 Processors)							
12	X'70'	X'2A'	X'02'	0	0	1	32767
12	X'70'	X'2B'	X'82'	0	0	1	32767
QTDSQLX86 (Intel 80X86 Processors)							
12	X'70'	X'2A'	X'02'	0	0	1	32767
12	X'70'	X'2B'	X'82'	0	0	1	32767
QTDSQLASC (IEEE ASCII Processors)							
12	X'70'	X'2A'	X'02'	0	0	1	32767
12	X'70'	X'2B'	X'82'	0	0	1	32767
QTDSQLVAX (VAX Processors)							
12	X'70'	X'2A'	X'02'	0	0	1	32767
12	X'70'	X'2B'	X'82'	0	0	1	32767
Example Descriptor in Hex							
(QTDSQL370 nullable form)				07780005	01022B		
				0C702B82	00000000	00017FFF	

Figure 5-70 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'02'	X'2C' (NTB) X'2D' (NNTB)
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	X'70'	X'2C'	X'03'	0	0	1 32767
12	X'70'	X'2D'	X'83'	0	0	1 32767
QTDSQL400 (AS/400 Processors)						
12	X'70'	X'2C'	X'03'	0	0	1 32767
12	X'70'	X'2D'	X'83'	0	0	1 32767
QTDSQLX86 (Intel 80X86 Processors)						
12	X'70'	X'2C'	X'03'	0	0	1 32767
12	X'70'	X'2D'	X'83'	0	0	1 32767
QTDSQLASC (IEEE ASCII Processors)						
12	X'70'	X'2C'	X'03'	0	0	1 32767
12	X'70'	X'2D'	X'83'	0	0	1 32767
QTDSQLVAX (VAX Processors)						
12	X'70'	X'2C'	X'03'	0	0	1 32767
12	X'70'	X'2D'	X'83'	0	0	1 32767
Example Descriptor in Hex (QTDSQL370 nullable form)						
				07780005	01022D	
				0C702D83	00000000	00017FFF

**Figure 5-71** DRDA Type X'2C,2D' SQL Type 460,461 NULL-TERMINATED BYTES

5.6.5.23 Null-Terminated 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'02'	X'2E' (NTCS) X'2F' (NNTCS)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Mode Siz	Fld	Length
0	1	2	3	4 5 6 7	8 9	10	11

QTDSQL370 (System/370 Processors)

12	X'70'	X'2E'	X'14'	00000-00500(e)	1	1	32767
12	X'70'	X'2F'	X'94'	00000-00500(e)	1	1	32767

QTDSQL400 (AS/400 Processors)

12	X'70'	X'2E'	X'14'	00000-00500(e)	1	1	32767
12	X'70'	X'2F'	X'94'	00000-00500(e)	1	1	32767

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'2E'	X'14'	00000-00850(e)	1	1	32767
12	X'70'	X'2F'	X'94'	00000-00850(e)	1	1	32767

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'2E'	X'14'	00000-00819(e)	1	1	32767
12	X'70'	X'2F'	X'94'	00000-00819(e)	1	1	32767

QTDSQLVAX (VAX Processors)

12	X'70'	X'2E'	X'14'	00000-00819(e)	1	1	32767
12	X'70'	X'2F'	X'94'	00000-00819(e)	1	1	32767

Example Descriptor in Hex    07780005 01022F  
 (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 5-72** DRDA Type X'2E,2F' SQL Type 460,461 NULL-TERMINATED SBCS

5.6.5.24 Fixed 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'02'	X'30' (FCS) X'31' (NFCS)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID				Chr Res-Siz	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 010231  
 (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 5-73 DRDA Type X'30,31' SQL Type 452,453 FIXED CHARACTER SBCS

5.6.5.25 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'02'	X'32' (VCS) X'33' (NVCS)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Mode Siz	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'33'	X'91'	00000-00500(e)	1	1	32767

QTDSQL400 (AS/400 Processors)

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)

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)

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)

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 010233  
 (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 5-74** 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'02'	X'34' (LVCS) X'35' (NLVCS)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Mode Siz	Fld	Length
0	1	2	3	4 5 6 7	8 9	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 010235  
 (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 5-75** DRDA Type X'34,35' SQL Type 456,457 LONG VAR CHARACTER SBCS

5.6.5.27 Fixed-Character DBCS (GRAPHIC)

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'02'	X'36' (FCD) X'37' (NFCD)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID				Chr Res-Siz	erved	Fld	Length
0	1	2	3	4	5	6	7	8	9	10	11

QTDSQL370 (System/370 Processors)

12	X'70'	X'36'	X'10'	00000-00300(f)				2	0	32767	
12	X'70'	X'37'	X'90'	00000-00300(f)				2	0	32767	

QTDSQL400 (AS/400 Processors)

12	X'70'	X'36'	X'10'	00000-00300(f)				2	0	32767	
12	X'70'	X'37'	X'90'	00000-00300(f)				2	0	32767	

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'36'	X'10'	00000-00301(f)				2	0	32767	
12	X'70'	X'37'	X'90'	00000-00301(f)				2	0	32767	

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'36'	X'10'	00000-01200(f)				2	0	32767	
12	X'70'	X'37'	X'90'	00000-01200(f)				2	0	32767	

QTDSQLVAX (VAX Processors)

12	X'70'	X'36'	X'10'	00000-01200(f)				2	0	32767	
12	X'70'	X'37'	X'90'	00000-01200(f)				2	0	32767	

Example Descriptor in Hex    07780005 010237  
 (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 5-76** DRDA Type X'36,37' SQL Type 468,469 FIXED CHARACTER DBCS

5.6.5.28 Variable-Character DBCS (GRAPHIC)

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'02'	X'38' (VCD) X'39' (NVCD)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Mode Siz	Fld	Length
0	1	2	3	4 5 6 7	8 9	10	11

QTDSQL370 (System/370 Processors)

12	X'70'	X'38'	X'11'	00000-00300(f)	2	1	32767
12	X'70'	X'39'	X'91'	00000-00300(f)	2	1	32767

QTDSQL400 (AS/400 Processors)

12	X'70'	X'38'	X'11'	00000-00300(f)	2	1	32767
12	X'70'	X'39'	X'91'	00000-00300(f)	2	1	32767

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'38'	X'11'	00000-00301(f)	2	1	32767
12	X'70'	X'39'	X'91'	00000-00301(f)	2	1	32767

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'38'	X'11'	00000-01200(f)	2	1	32767
12	X'70'	X'39'	X'91'	00000-01200(f)	2	1	32767

QTDSQLVAX (VAX Processors)

12	X'70'	X'38'	X'11'	00000-01200(f)	2	1	32767
12	X'70'	X'39'	X'91'	00000-01200(f)	2	1	32767

Example Descriptor in Hex 07780005 010239  
 (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 5-77 DRDA Type X'38,39' SQL Type 464,465 VARIABLE CHARACTER DBCS

5.6.5.29 Long Variable Character DBCS (GRAPHIC)

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'02'	X'3A' (LVCD) X'3B' (NLVCD)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Mode Siz	Fld	Length
0	1	2	3	4 5 6 7	8 9	10	11

QTDSQL370 (System/370 Processors)

12	X'70'	X'3A'	X'11'	00000-00300(f)	2	1	32767
12	X'70'	X'3B'	X'91'	00000-00300(f)	2	1	32767

QTDSQL400 (AS/400 Processors)

12	X'70'	X'3A'	X'11'	00000-00300(f)	2	1	32767
12	X'70'	X'3B'	X'91'	00000-00300(f)	2	1	32767

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'3A'	X'11'	00000-00301(f)	2	1	32767
12	X'70'	X'3B'	X'91'	00000-00301(f)	2	1	32767

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'3A'	X'11'	00000-01200(f)	2	1	32767
12	X'70'	X'3B'	X'91'	00000-01200(f)	2	1	32767

QTDSQLVAX (VAX Processors)

12	X'70'	X'3A'	X'11'	00000-01200(f)	2	1	32767
12	X'70'	X'3B'	X'91'	00000-01200(f)	2	1	32767

Example Descriptor in Hex    07780005 01023B  
 (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 5-78** DRDA Type X'3A,3B' SQL Type 472,473 LONG VAR CHARACTER DBCS

5.6.5.30 Fixed Character 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'02'	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 01023D  
 (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 5-79** DRDA Type X'3C,3D' SQL Type 452,453 FIXED CHARACTER MIXED

5.6.5.31 Variable Character 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'02'	X'3E' (VCM) X'3F' (NVCM)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Mode Siz	Fld	Length
0	1	2	3	4 5 6 7	8 9	10	11

QTDSQL370 (System/370 Processors)

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)

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)

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)

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)

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 01023F  
 (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 5-80** DRDA Type X'3E,3F' SQL Type 448,449 VARIABLE CHARACTER MIXED

5.6.5.32 Long Variable Character 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'02'	X'40' (LVCM) X'41' (NLVCM)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Mode Siz	Fld	Length
0	1	2	3	4 5 6 7	8 9	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 010241  
 (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 5-81** 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'02'	X'42' (NTM) X'43' (NNTM)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Mode Siz	Fld	Length
0	1	2	3	4 5 6 7	8 9	10	11

QTDSQL370 (System/370 Processors)

12	X'70'	X'42'	X'14'	00000-00930(g)	1	1	32767
12	X'70'	X'43'	X'94'	00000-00930(g)	1	1	32767

QTDSQL400 (AS/400 Processors)

12	X'70'	X'42'	X'14'	00000-00930(g)	1	1	32767
12	X'70'	X'43'	X'94'	00000-00930(g)	1	1	32767

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'42'	X'14'	00000-00932(g)	1	1	32767
12	X'70'	X'43'	X'94'	00000-00932(g)	1	1	32767

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'42'	X'14'	00000-01200(g)	1	1	32767
12	X'70'	X'43'	X'94'	00000-01200(g)	1	1	32767

QTDSQLVAX (VAX Processors)

12	X'70'	X'42'	X'14'	00000-01200(g)	1	1	32767
12	X'70'	X'43'	X'94'	00000-01200(g)	1	1	32767

Example Descriptor in Hex 07780005 010243  
 (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 5-82** DRDA Type X'42,43' SQL Type 460,461 NULL-TERMINATED MIXED

5.6.5.34 Pascal L String 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'02'	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	X'70'	X'44'	X'07'	0	0	1
12	X'70'	X'45'	X'87'	0	0	1
						255
						255
QTDSQL400 (AS/400 Processors)						
12	X'70'	X'44'	X'07'	0	0	1
12	X'70'	X'45'	X'87'	0	0	1
						255
						255
QTDSQLX86 (Intel 80X86 Processors)						
12	X'70'	X'44'	X'07'	0	0	1
12	X'70'	X'45'	X'87'	0	0	1
						255
						255
QTDSQLASC (IEEE ASCII Processors)						
12	X'70'	X'44'	X'07'	0	0	1
12	X'70'	X'45'	X'87'	0	0	1
						255
						255
QTDSQLVAX (VAX Processors)						
12	X'70'	X'44'	X'07'	0	0	1
12	X'70'	X'45'	X'87'	0	0	1
						255
						255

Example Descriptor in Hex    07780005 010245  
 (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 5-83 DRDA Type X'44,45' SQL Type 476,477 PASCAL L STRING BYTES

5.6.5.35 Pascal L String 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'02'	X'46' (PLS) X'47' (NPLS)	
FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Mode Siz	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 010247							
(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 5-84** DRDA Type X'46,47' SQL Type 476,477 PASCAL L STRING SBCS

5.6.5.36 Pascal L String 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'02'	X'48' (PLM) X'49' (NPLM)	
FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Mode Siz	Fld Length	
0	1	2	3	4 5 6 7	8 9	10 11	
QTDSQL370 (System/370 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
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 010249							
(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 5-85 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'02'	X'4C' (DLS) X'4D' (NDLS)
---	--------------	---	-----------------	--------------------	--------------------	-----------------------------

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Mode Siz	Fld	Length
0	1	2	3	4 5 6 7	8 9	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    01024D  
 (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 5-86** DRDA Type X'4C,4D' SQL Type 396,397 SBCS DATALINK

5.6.5.38 Mixed-Byte 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'02'	X'4E' (DLM) X'4F' (NDLM)		
FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Mode Siz	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	01024F			
(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 5-87 DRDA Type X'4E,4F' SQL Type 396,397 MIXED-BYTE DATALINK

5.6.5.39 Decimal Floating Point

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Type 5	DRDA Type 6
Meta Data (Environment-independent)						
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'02'	X'BA' (DFP) X'BB' (NDFP)
FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Mode Siz	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'BA' X'BB'	X'42' X'C2'	0 0	0 0	16 or 8 16 or 8
QTDSQL400 (AS/400 Processors)						
12 12	X'70' X'70'	X'BA' X'BB'	X'42' X'C2'	0 0	0 0	16 or 8 16 or 8
QTDSQLX86 (Intel 80X86 Processors)						
12 12	X'70' X'70'	X'BA' X'BB'	X'42' X'C2'	0 0	0 0	16 or 8 16 or 8
QTDSQLASC (IEEE ASCII Processors)						
12 12	X'70' X'70'	X'BA' X'BB'	X'42' X'C2'	0 0	0 0	16 or 8 16 or 8
QTDSQLVAX (VAX Processors)						
12 12	X'70' X'70'	X'BA' X'BB'	X'42' X'C2'	0 0	0 0	16 or 8 16 or 8
Example Descriptor in Hex (QTDSQL370 nullable form)						
				07780005	0102BB	
				0C70BBC2	00000000	00000010

Figure 5-88 DRDA Type X'BA, BB' SQL Type 996, 997 DECFLOAT (34)

5.6.5.40 Boolean

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Type 5	DRDA Type 6	
Meta Data (Environment-independent)							
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'02'	X'BE' (BL) X'BF' (NBL)	
FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	Reserved	Reserved	Fld Length	
0	1	2	3	4 5 6 7	8 9	10 11	
QTDSQL370 (System/370 Processors)							
12	X'70'	X'BE'	X'25'	0	0	0	2
12	X'70'	X'BF'	X'A5'	0	0	0	2
QTDSQL400 (AS/400 Processors)							
12	X'70'	X'BE'	X'25'	0	0	0	2
12	X'70'	X'BF'	X'A5'	0	0	0	2
QTDSQLX86 (Intel 80X86 Processors)							
12	X'70'	X'BE'	X'25'	0	0	0	2
12	X'70'	X'BF'	X'A5'	0	0	0	2
QTDSQLASC (IEEE ASCII Processors)							
12	X'70'	X'BE'	X'25'	0	0	0	2
12	X'70'	X'BF'	X'A5'	0	0	0	2
QTDSQLVAX (VAX Processors)							
12	X'70'	X'BE'	X'25'	0	0	0	2
12	X'70'	X'BF'	X'A5'	0	0	0	2
Example Descriptor in Hex (QTDSQL370 nullable form)							
				07780005	0102BF		
				0C70BFA5	00000000	00000002	

Figure 5-89 DRDA Type X'BE,BF' SQL Type 2436,2437 BOOLEAN

5.6.5.41 Fixed Binary

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Type 5	DRDA Type 6	
Meta Data (Environment-independent)							
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'02'	X'C0' (FBIN) X'C1' (NFBIN)	
FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Mode Siz	Fld Length	
0	1	2	3	4 5 6 7	8 9	10 11	
QTDSQL370 (System/370 Processors)							
12	X'70'	X'C0'	X'01'	0	0	0	32767
12	X'70'	X'C1'	X'81'	0	0	0	32767
QTDSQL400 (AS/400 Processors)							
12	X'70'	X'C0'	X'01'	0	0	0	32767
12	X'70'	X'C1'	X'81'	0	0	0	32767
QTDSQLX86 (Intel 80X86 Processors)							
12	X'70'	X'C0'	X'01'	0	0	0	32767
12	X'70'	X'C1'	X'81'	0	0	0	32767
QTDSQLASC (IEEE ASCII Processors)							
12	X'70'	X'C0'	X'01'	0	0	0	32767
12	X'70'	X'C1'	X'81'	0	0	0	32767
QTDSQLVAX (VAX Processors)							
12	X'70'	X'C0'	X'01'	0	0	0	32767
12	X'70'	X'C1'	X'81'	0	0	0	32767
Example Descriptor in Hex (QTDSQL370 nullable form)							
				07780005	0102C1		
				0C70C181	00000000	00007FFF	

Figure 5-90 DRDA Type X'C0,C1' SQL Type 912,913 FIXED BINARY

5.6.5.42 Variable Binary

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Type 5	DRDA Type 6	
Meta Data (Environment-independent)							
7	MDD X'78'	0	Rel.DB X'05'	Data Type X'01'	Next Byte X'02'	X'C2' (VBIN X'C3' (NVBIN)	
FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Mode Siz	Fld Length	
0	1	2	3	4 5 6 7	8 9	10 11	
QTDSQL370 (System/370 Processors)							
12	X'70'	X'C2'	X'02'	0	0	1	32767
12	X'70'	X'C3'	X'82'	0	0	1	32767
QTDSQL400 (AS/400 Processors)							
12	X'70'	X'C2'	X'02'	0	0	1	32767
12	X'70'	X'C3'	X'82'	0	0	1	32767
QTDSQLX86 (Intel 80X86 Processors)							
12	X'70'	X'C2'	X'02'	0	0	1	32767
12	X'70'	X'C3'	X'82'	0	0	1	32767
QTDSQLASC (IEEE ASCII Processors)							
12	X'70'	X'C2'	X'02'	0	0	1	32767
12	X'70'	X'C3'	X'82'	0	0	1	32767
QTDSQLVAX (VAX Processors)							
12	X'70'	X'C2'	X'02'	0	0	1	32767
12	X'70'	X'C3'	X'82'	0	0	1	32767
Example Descriptor in Hex (QTDSQL370 nullable form)							
				07780005	0102C3		
				0C70C382	00000000	00017FFF	

Figure 5-91 DRDA Type X'C2,C3' SQL Type 908,909 VARIABLE BINARY

5.6.5.43 XML String Internal Encoding

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Type 5	DRDA Type 6
Meta Data (Environment-independent)						
7	MDD X'78'	0	X'05'	X'01'	X'02'	X'C4' (XMLSIE) X'C5' (NXMLSIE)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID 4	5	6	7	Chr Mode Siz 8	9	Fld 10	Length 11
0	1	2	3								

QTDSQL370 (System/370 Processors)

12	X'70'	X'C4'	X'50'	0	0	0	8(h)
12	X'70'	X'C5'	X'D0'	0	0	0	8(h)

QTDSQL400 (AS/400 Processors)

12	X'70'	X'C4'	X'50'	0	0	0	8(h)
12	X'70'	X'C5'	X'D0'	0	0	0	8(h)

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'C4'	X'50'	0	0	0	8(h)
12	X'70'	X'C5'	X'D0'	0	0	0	8(h)

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'C4'	X'50'	0	0	0	8(h)
12	X'70'	X'C5'	X'D0'	0	0	0	8(h)

QTDSQLVAX (VAX Processors)

12	X'70'	X'C4'	X'50'	0	0	0	8(h)
12	X'70'	X'C5'	X'D0'	0	0	0	8(h)

Example Descriptor in Hex 07780005 0102C5  
 (QTDSQL370 nullable form) 0C70C5D0 0000000000 00008008

- (h) The placeholder indicator bit is set to '1'B.
- (x) XML String Internal Encoding data conforms to DRDA Rule Dt22.

**Figure 5-92** DRDA Type X'C4,C5' SQL Type 988,989 XML

5.6.5.44 XML String External Encoding

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Type 5	DRDA Type 6
Meta Data (Environment-independent)						
7	MDD X'78'	0	X'05'	X'01'	X'02'	X'C6' (XMLSEE) X'C7' (NXMLSEE)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Mode Siz	Fld	Length
0	1	2	3	4 5 6 7	8 9	10	11
QTDSQL370 (System/370 Processors)							

12	X'C6'	0	X'51'	00000-01280(g)	1	0	8(h)
12	X'C7'	0	X'D1'	00000-01280(g)	1	0	8(h)

QTDSQL400 (AS/400 Processors)

12	X'C6'	0	X'51'	00000-01280(g)	1	0	8(h)
12	X'C7'	0	X'D1'	00000-01280(g)	1	0	8(h)

QTDSQLX86 (Intel 80X86 Processors)

12	X'C6'	0	X'51'	00000-01280(g)	1	0	8(h)
12	X'C7'	0	X'D1'	00000-01280(g)	1	0	8(h)

QTDSQLASC (IEEE ASCII Processors)

12	X'C6'	0	X'51'	00000-01280(g)	1	0	8(h)
12	X'C7'	0	X'D1'	00000-01280(g)	1	0	8(h)

QTDSQLVAX (VAX Processors)

12	X'C6'	0	X'51'	00000-01280(g)	1	0	8(h)
12	X'C7'	0	X'D1'	00000-01280(g)	1	0	8(h)

Example Descriptor in Hex 07780005 0102C7  
 (QTDSQL370 nullable form) 0C70C7D1 000004B8 01008008

- (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.
- (x) XML String External Encoding data conforms to DRDA Rule DT22 and Dt23.

**Figure 5-93** DRDA Type X'C6,C7' SQL Type 988,989 XML

5.6.5.45 Large Object 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'02'	X'C8' (OB) X'C9' (NOB)

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'C8'	X'50'	0	0	0	8 (h)
12	X'70'	X'C9'	X'D0'	0	0	0	8 (h)

QTDSQL400 (AS/400 Processors)

12	X'70'	X'C8'	X'50'	0	0	0	8 (h)
12	X'70'	X'C9'	X'D0'	0	0	0	8 (h)

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'C8'	X'50'	0	0	0	8 (h)
12	X'70'	X'C9'	X'D0'	0	0	0	8 (h)

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'C8'	X'50'	0	0	0	8 (h)
12	X'70'	X'C9'	X'D0'	0	0	0	8 (h)

QTDSQLVAX (VAX Processors)

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 0102C9  
 (QTDSQL370 nullable form)    0C70C9D0 00000000 00018008

(h) The placeholder indicator bit is set to `1'B.

**Figure 5-94** DRDA Type X'C8,C9' SQL Type 404,405 LARGE OBJECT BYTES

5.6.5.46 Large Object Character SBCS

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'02'	X'CA' (OCS) X'CB' (NOCS)

Meta Data (Environment-independent)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Siz	Rsvd	Fld	Length
0	1	2	3	4 5 6 7	8	9	10	11

QTDSQL370 (System/370 Processors)

12	X'70'	X'CA'	X'51'	00000-00500(e)	1	0	8 (h)
12	X'70'	X'CB'	X'D1'	00000-00500(e)	1	0	8 (h)

QTDSQL400 (AS/400 Processors)

12	X'70'	X'CA'	X'51'	00000-00500(e)	1	0	8 (h)
12	X'70'	X'CB'	X'D1'	00000-00500(e)	1	0	8 (h)

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'CA'	X'51'	00000-00850(e)	1	0	8 (h)
12	X'70'	X'CB'	X'D1'	00000-00850(e)	1	0	8 (h)

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'CA'	X'51'	00000-00819(e)	1	0	8 (h)
12	X'70'	X'CB'	X'D1'	00000-00819(e)	1	0	8 (h)

QTDSQLVAX (VAX Processors)

12	X'70'	X'CA'	X'51'	00000-00819(e)	1	0	8 (h)
12	X'70'	X'CB'	X'D1'	00000-00819(e)	1	0	8 (h)

Example Descriptor in Hex 07780005 0102CB  
 (QTDSQL370 nullable form) 0C70CBD1 000001F4 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 5-95 DRDA Type X'CA, CB' SQL Type 408,409 LARGE OBJECT CHAR. SBCS

5.6.5.47 Large Object Character DBCS (GRAPHIC)

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'02'	X'CC' (OCD) X'CD' (NOCD)

Meta Data (Environment-independent)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Siz	Rsvd	Fld	Length
0	1	2	3	4 5 6 7	8	9	10	11

QTDSQL370 (System/370 Processors)

12	X'70'	X'CC'	X'51'	00000-00300(f)	2	0	8 (h)
12	X'70'	X'CD'	X'D1'	00000-00300(f)	2	0	8 (h)

QTDSQL400 (AS/400 Processors)

12	X'70'	X'CC'	X'51'	00000-00300(f)	2	0	8 (h)
12	X'70'	X'CD'	X'D1'	00000-00300(f)	2	0	8 (h)

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'CC'	X'51'	00000-00301(f)	2	0	8 (h)
12	X'70'	X'CD'	X'D1'	00000-00301(f)	2	0	8 (h)

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'CC'	X'51'	00000-01200(f)	2	0	8 (h)
12	X'70'	X'CD'	X'D1'	00000-01200(f)	2	0	8 (h)

QTDSQLVAX (VAX Processors)

12	X'70'	X'CC'	X'51'	00000-01200(f)	2	0	8 (h)
12	X'70'	X'CD'	X'D1'	00000-01200(f)	2	0	8 (h)

Example Descriptor in Hex 07780005 0102CD  
 (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 5-96** DRDA Type X'CC,CD' SQL Type 412,413 LARGE OBJECT CHAR. DBCS

5.6.5.48 Large Object Character Mixed

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'02'	X'CE' (OCM) X'CF' (NOCM)

Meta Data (Environment-independent)

FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	CCSID	Chr Siz	Rsvd	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	0	8 (h)
12	X'70'	X'CF'	X'D1'	00000-00930(g)	1	0	8 (h)

QTDSQL400 (AS/400 Processors)

12	X'70'	X'CE'	X'51'	00000-00930(g)	1	0	8 (h)
12	X'70'	X'CF'	X'D1'	00000-00930(g)	1	0	8 (h)

QTDSQLX86 (Intel 80X86 Processors)

12	X'70'	X'CE'	X'51'	00000-00932(g)	1	0	8 (h)
12	X'70'	X'CF'	X'D1'	00000-00932(g)	1	0	8 (h)

QTDSQLASC (IEEE ASCII Processors)

12	X'70'	X'CE'	X'51'	00000-01200(g)	1	0	8 (h)
12	X'70'	X'CF'	X'D1'	00000-01200(g)	1	0	8 (h)

QTDSQLVAX (VAX Processors)

12	X'70'	X'CE'	X'51'	00000-01200(g)	1	0	8 (h)
12	X'70'	X'CF'	X'D1'	00000-01200(g)	1	0	8 (h)

Example Descriptor in Hex 07780005 0102CF  
(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 5-97** 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 5-98](#) is the standard representation for this information in this environment. (This is taken from [Section 5.6.5.24](#) (on page 360).)

Length 0	Type 1	Identity 2	Class 3	MD Type 4	MD Ref Type 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'	DRDA Type X'30' (FCS)	
FD Tr Ln	FD:OCA Tripl 'SDA'	FD:OCA Tripl LID	FD:OCA Field Type	Res- erved	CCSID	Chr Mode Siz	Prec/Scale or Fld Length
0	1	2	3	4 5	6 7	8 9	10 11
QTDSQLX86 (Intel 80X86 Processors)							
12	X'70'	X'30'	X'10'	00000-00850(e)	1	0	32767

Example Descriptor in Hex    07780005 010230  
 (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 5-98** DRDA Type X'30', SQL Type 468, MDD Override Example: Base

The definition in [Figure 5-99](#) specifies the other character set needed to properly represent the box drawing character data.

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'	DRDA Type X'30' (FCS)	
FD Tr Ln 0	FD:OCA Tripl 'SDA' 1	FD:OCA Tripl LID 2	FD:OCA Field Type 3	Res- erved 4	CCSID 5	Chr Siz 8	Prec;/Scale or Fld Length 10 11
QTDSQLX86 (Intel 80X86 Processors)							
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 5-99** 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 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 five meta data types. These types are:

1. Relate DRDA and SQL data types to their representations.
2. Relate names of simple 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.
5. Relate names of complex group 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 5-11** 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'02'	X'02'(I4)	496	4-byte Integer
X'05'	X'02'	X'03'(NI4)	497	Nullable 4-byte Integer
X'05'	X'02'	X'04'(I2)	500	2-byte Integer
X'05'	X'02'	X'05'(NI2)	501	Nullable 2-byte Integer
X'05'	X'02'	X'06'(I1)	n/a	1-byte Integer
X'05'	X'02'	X'07'(NI1)	n/a	Nullable 1-byte Integer
X'05'	X'02'	X'08'(BF16)	(480)	16-byte Basic Floating Point
X'05'	X'02'	X'09'(NBF16)	(481)	Nullable 16-byte Basic Floating Point
X'05'	X'02'	X'0A'(BF8)	480	8-byte Basic Floating Point
X'05'	X'02'	X'0B'(NBF8)	481	Nullable 8-byte Basic Floating Point
X'05'	X'02'	X'0C'(BF4)	480	4-byte Basic Floating Point
X'05'	X'02'	X'0D'(NBF4)	481	Nullable 4-byte Basic Floating Point
X'05'	X'02'	X'0E'(FD)	484	Fixed Decimal
X'05'	X'02'	X'0F'(NFD)	485	Nullable Fixed Decimal
X'05'	X'02'	X'10'(ZD)	488	Zoned Decimal
X'05'	X'02'	X'11'(NZD)	489	Nullable Zoned Decimal
X'05'	X'02'	X'12'(N)	504	Numeric Character
X'05'	X'02'	X'13'(NN)	505	Nullable Numeric Character
X'05'	X'02'	X'14'(RSL)	972	Result Set Locator
X'05'	X'02'	X'15'(NRSL)	973	Nullable Result Set Locator

Application Class	Meta Data Type	Meta Data Reference DRDA-Type and Name	SQL Data Type	Description
X'05'	X'02'	X'16'(I8)	492	Eight-byte Integer
X'05'	X'02'	X'17'(NI8)	493	Nullable Eight-byte Integer
X'05'	X'02'	X'18'(OBL)	960	Large Object Bytes Locator
X'05'	X'02'	X'19'(NOBL)	961	Nullable Large Object Bytes Locator
X'05'	X'02'	X'1A'(OCL)	964	Large Object Character Locator
X'05'	X'02'	X'1B'(NOCL)	965	Nullable Large Object Character Locator
X'05'	X'02'	X'1C'(OCDL)	968	Large Object Character DBCS Locator
X'05'	X'02'	X'1D'(NOCDL)	969	Nullable Large Obj. Char. DBCS Locator
X'05'	X'02'	X'1E'(RI)	904	Row Identifier
X'05'	X'02'	X'1F'(NRI)	905	Nullable Row Identifier
X'05'	X'02'	X'20'(D)	384	Date
X'05'	X'02'	X'21'(ND)	385	Nullable Date
X'05'	X'02'	X'22'(T)	388	Time
X'05'	X'02'	X'23'(NT)	389	Nullable Time
X'05'	X'02'	X'24'(TS)	392	Timestamp
X'05'	X'02'	X'25'(NTS)	393	Nullable Timestamp
X'05'	X'02'	X'26'(FB)	452	Fixed Bytes
X'05'	X'02'	X'27'(NFB)	453	Nullable Fixed Bytes
X'05'	X'02'	X'28'(VB)	448	Variable Bytes
X'05'	X'02'	X'29'(NVB)	449	Nullable Variable Bytes
X'05'	X'02'	X'2A'(LVB)	456	Long Variable Bytes
X'05'	X'02'	X'2B'(NLVB)	457	Nullable Long Variable Bytes
X'05'	X'02'	X'2C'(NTB)	460	Null-Terminated Bytes
X'05'	X'02'	X'2D'(NNTB)	461	Nullable Null-Terminated Bytes
X'05'	X'02'	X'2E'(NTCS)	460	Null-Terminated SBCS
X'05'	X'02'	X'2F'(NNTCS)	461	Nullable Null-Terminated SBCS
X'05'	X'02'	X'30'(FCS)	452	Fixed Character SBCS
X'05'	X'02'	X'31'(NFCS)	453	Nullable Fixed Character SBCS
X'05'	X'02'	X'32'(VCS)	448	Variable Character SBCS
X'05'	X'02'	X'33'(NVCS)	449	Nullable Variable Character SBCS
X'05'	X'02'	X'34'(LVCS)	456	Long Variable Character SBCS
X'05'	X'02'	X'35'(NLVCS)	457	Nullable Long Variable Character SBCS
X'05'	X'02'	X'36'(FCD)	468	Fixed Character DBCS
X'05'	X'02'	X'37'(NFCD)	469	Nullable Fixed Character DBCS
X'05'	X'02'	X'38'(VCD)	464	Variable Character DBCS
X'05'	X'02'	X'39'(NVCD)	465	Nullable Variable Character DBCS
X'05'	X'02'	X'3A'(LVCD)	472	Long Variable Character DBCS
X'05'	X'02'	X'3B'(NLVCD)	473	Nullable Long Variable Character DBCS
X'05'	X'02'	X'3C'(FCM)	452	Fixed Character Mixed
X'05'	X'02'	X'3D'(NFCM)	453	Nullable Fixed Character Mixed
X'05'	X'02'	X'3E'(VCM)	448	Variable Character Mixed
X'05'	X'02'	X'3F'(NVCM)	449	Nullable Variable Character Mixed
X'05'	X'02'	X'40'(LVCM)	456	Long Variable Character Mixed
X'05'	X'02'	X'41'(NLVCM)	457	Nullable Long Variable Character Mixed
X'05'	X'02'	X'42'(NTM)	460	Null-Terminated Mixed
X'05'	X'02'	X'43'(NNTM)	461	Nullable Null-Terminated Mixed
X'05'	X'02'	X'44'(PLB)	476	Pascal L String Bytes
X'05'	X'02'	X'45'(NPLB)	477	Nullable Pascal L String Bytes

Application Class	Meta Data Type	Meta Data Reference DRDA-Type and Name	SQL Data Type	Description
X'05'	X'02'	X'46'(PLS)	476	Pascal L String SBCS
X'05'	X'02'	X'47'(NPLS)	477	Nullable Pascal L String SBCS
X'05'	X'02'	X'48'(PLM)	476	Pascal L String Mixed
X'05'	X'02'	X'49'(NPLM)	477	Nullable Pascal L String Mixed
X'05'	X'02'	X'4C'(DLS)	396	SBCS Datalink
X'05'	X'02'	X'4D'(NDLS)	397	Nullable SBCS Datalink
X'05'	X'02'	X'4E'(DLM)	396	Mixed-byte Datalink
X'05'	X'02'	X'4F'(NDLM)	396	Nullable Mixed-byte Datalink
X'05'	X'02'	X'BA'(DFP)	996	Decimal Floating Point
X'05'	X'02'	X'BB'(NDFP)	997	Nullable Decimal Floating Point
X'05'	X'02'	X'BE'(BL)	2436	Boolean
X'05'	X'02'	X'BF'(NBL)	2437	Nullable Boolean
X'05'	X'02'	X'C0'(FBIN)	912	Fixed Binary
X'05'	X'02'	X'C1'(NFBIN)	913	Nullable Fixed Binary
X'05'	X'02'	X'C2'(VBIN)	908	Variable Binary
X'05'	X'02'	X'C3'(NVBIN)	909	Nullable Variable Binary
X'05'	X'02'	X'C4'(XMLSIE)	988	XML String Internal Encoding
X'05'	X'02'	X'C5'(NXMLSIE)	989	Nullable XML String Internal Encoding
X'05'	X'02'	X'C6'(XMLSEE)	988	XML String External Encoding
X'05'	X'02'	X'C7'(NXMLSEE)	989	Nullable XML String External Encoding
X'05'	X'02'	X'C8'(OB)	404	Large Object Bytes
X'05'	X'02'	X'C9'(NOB)	405	Nullable Large Object Bytes
X'05'	X'02'	X'CA'(OCS)	408	Large Object Character SBCS
X'05'	X'02'	X'CB'(NOCS)	409	Nullable Large Object Character SBCS
X'05'	X'02'	X'CC'(OCD)	412	Large Object Character DBCS
X'05'	X'02'	X'CD'(NOCD)	413	Nullable Large Object Character DBCS
X'05'	X'02'	X'CE'(OCM)	408	Large Object Character Mixed
X'05'	X'02'	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 5-12** 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'D1'	SQLDIAGGRP	SQL Diagnostic Group
X'05'	X'02'	X'D3'	SQLDIAGSTT	SQL Diagnostic Statement Group
X'05'	X'02'	X'D5'	SQLDDCGRP	SQL Diagnostic Condition Group
X'05'	X'02'	X'D6'	SQLCNGRP	SQL Diagnostic Connection Group
X'05'	X'02'	X'D7'	SQLTOKGRP	SQL Diagnostic Token Group
X'05'	X'02'	X'D8'	SQLDCXGRP	SQL Diagnostic External Name Group

**Table 5-13** 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'E5'	SQLDCROW	SQL Diagnostic Condition Row
X'05'	X'03'	X'E6'	SQLCNROW	SQL Diagnostic Connection Row
X'05'	X'03'	X'E7'	SQLTOKROW	SQL Diagnostic Token Row

**Table 5-14** 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'F5'	SQLDIAGCI	SQL Diagnostic Condition Array
X'05'	X'04'	X'F6'	SQLDIAGCN	SQL Diagnostic Connection Array
X'05'	X'04'	X'F7'	SQLDCTOKS	SQL Condition Token Array

**Table 5-15** 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 5-16** 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

**Table 5-17** 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



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. The values are the same for the early and late descriptors. They are established by specifying TYPDEFNAM and/or TYPDEFOVR. The M triplets define DRDA information units (such as SQLCA). The defaults are established with the MGRLVL parameter on EXCSAT. The defaults may be overridden at subsequent points in processing by the MGRLVLOVR object (see [Section 4.3.5](#) (on page 85)). The T triplet values can be overridden by a late descriptor 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 392).) In some cases, TYPDEFNAM and TYPDEFOVR can be specified to override the representation specification provided on the earlier ACCRDB command.

[Table 5-19](#) illustrates the cases:

**Table 5-19** 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 three groups: the T triplets, the O triplets, and the U triplets. The O triplets provide specific overrides and are described in [Section 5.7.1.2](#) (on page 392). The T triplet values establish the basic representations for all DRDA data. The values are the same for the early and late descriptors. 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 T and U 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 U 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).

Late LID assignments are as follows:

Late												
.....												
Environmental				Overrides					Environmental		Grp Row Arr	
TTTTTTTTTTTTTTTT	OOOOOOOOOOOOOOOO					TTTTTTT	UUUUUUUUUUUUUU					
0x 1x 2x 3x 4x	5x 6x 7x 8x 9x Ax	Bx Cx	Dx Ex Fx									

Observations when assigning O triplet LIDs:

1. The O triplet LIDs have space reserved from X'50' to X'AF'. 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 128) 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.

5. There are cases when a late MDD (MD Ref Type=X'01') reference an early MDD (MD Ref Type=X'02'). For example, the late SQLCADTA row descriptor uses the early SQLCAGRP group descriptor to describe an SQLCA. An O triplet only references user data and does not interfere with early M triplets. A late O triplet only references late M and T triplets and does not block references to early descriptors. Thus, if X'54' is assigned an O triplet which is also the value assigned to the early SQLCAGRP group, the O triplet does not prevent the use of the M triplet for the SQLCAGRP in the SQLCADTA.

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 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 default early environment descriptor set is determined during the Access RDB phase of communication establishment between requester and server. The default early data unit descriptor set is determined during EXCSAT based on the SQLAM's MGRLVL.<sup>47</sup> 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 5-20](#) 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 Late Environmental Descriptors

The Environmental Descriptors in [Table 5-20](#) apply to both early and late descriptors.

**Table 5-20** Complete Set of Late 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
NN	07780005 010113	0C7013B2 000001F4 01001F1F

47. Defaults may be overridden after connection by means of the TYPDEFOVR, TYPDEFNAM, and MGRLVLOVR objects. Refer to [Section 7.9](#) and [Section 4.3.5](#) for details.

DRDA Type	MDD Descriptor—HEX	SDA, GDA, CPT, or RLO Descriptor—HEX
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	07780005 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
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

DRDA Type	MDD Descriptor—HEX	SDA, GDA, CPT, or RLO Descriptor—HEX
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
DFP	07780005 0101BA	0C70BA42 00000000 00000010 or 0C70BA42 00000000 00000008
NDFP	07780005 0101BB	0C70BBC2 00000000 00000010 or 0C70BBC2 00000000 00000008
BL	07780005 0101BE	0C70BE25 00000000 00000002
NBL	07780005 0101BF	0C70BFA5 00000000 00000002
FBIN	07780005 0101C0	0C70C001 00000000 00007FFF
NFBIN	07780005 0101C1	0C70C181 00000000 00007FFF
VBIN	07780005 0101C2	0C70C202 00000000 00017FFF
NVBIN	07780005 0101C3	0C70C382 00000000 00017FFF
XMLSIE	07780005 0101C4	0C70C450 0000000000 00008008
NXMLSIE	07780005 0101C5	0C70C5D0 0000000000 00008008
XMLSEE	07780005 0101C6	0C70C651 000004B8 01008008
NXMLSEE	07780005 0101C7	0C70C7D1 000004B8 01008008
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 5-21](#) 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 5-21** Complete Set of Early Data Unit Group Descriptors

DRDA Type	MDD Descriptor in HEX	SDA, GDA, CPT, or RLO Descriptor in HEX
SQLDAGRP	07780005020250	157550 040002 040002 160008 040002 260002 D20000
SQLUDTGRP	07780005020251	157651 020004 3200FF 3E00FF 3200FF 3E00FF 3200FF
SQLCAXGRP	07780005020252	3F7652 020004 020004 020004 020004 020004 020004 300001 300001 300001 300001 300001 300001 300001 300001 300001 300001 300001 3200FF 3E0046 320046
SQLCAGRP	07780005020254	127654 020004 300005 300008 520000 560000
SQLARYGRP	07780005020256	097556 020004 020004

DRDA Type	MDD Descriptor in HEX	SDA, GDA, CPT, or RLO Descriptor in HEX
SQLNUMGRP	07780005020258	067558 040002
SQLOBJGRP	0778000502025A	09755A 3E00FF 3200FF
SQLSTTGRP	0778000502025C	09755C CF0004 CB0004
SQLVRBGRP	0778000502025E	21755E 040002 040002 160008 040002 260002 3E00FF 3200FF 3E00FF 3200FF 5B0000
SQLRSGRP	0778000502025F	0F755F 140004 3E00FF 3200FF 020004
SQLDAROW	07780005030260	067160 500001
SQLCARD	07780005030264	067164 540001
SQLARYROW	07780005030266	067166 560001
SQLNUMROW	07780005030268	067558 040002
SQLOBJNAM	0778000503026A	06716A 5A0001
SQLSTT	0778000503026C	06716C 5C0001
SQLVRBROW	0778000503026E	06716E 5E0001
SQLRSROW	0778000503026F	06716F 5F0001
SQLDARD	07780005040174	0F7174 640001 E00001 680001 600000
SQLNUMEXT	07780005040276	067176 660001
SQLCINRD	0778000504027B	0C717B E00001 680001 6F0000
SQLSTTVRB	0778000504027E	09717E 680001 6E0000
SQLRSLRD	0778000504027F	0971F6 680001 6F0000
SQLDHGRP	077800050202D0	1E76D0 040002 040002 040002 040002 040002 040002 3200FF 3E00FF 3200FF
SQLDIAGGRP	077800050202D1	0C76D1 D30000 F50000 F60000
SQLDOPTGRP	077800050202D2	1E76D2 040002 3E00FF 3200FF 3E00FF 3200FF 3E00FF 3200FF 5B0000 D40000
SQLDIAGSTT	077800050202D3	3376D3 020004 020004 020004 020004 020004 020004 020004 1 60008 160008 160008 300001 300001 300001 300001 300001 300001 300001 300001
SQLDXGRP	077800050202D4	2A76D4 040002 040002 040002 040002 3200FF 3200FF 3200FF 3200FF 3200FF 3E00FF 3200FF 3E00FF 3200FF
SQLDCGRP	077800050202D5	4E75D5 020004 300005 020004 020004 160008 020004 020004 020004 020004 020004 020004 30000 A 300008 300005 F70000 3F7FA0 337FA0 3F00FF 3300FF 3F00FF 3300FF 3F00FF 3300FF D30001
SQLCNGRP	077800050202D6	1B75D6 020004 020004 300001 300001 300008 3200 FF 3200FF 3200FF
SQLTOKGRP	077800050302D7	0971D7 3F00FF 3300FF
SQLDCXGRP	077800050202D8	4576D8 3200FF 3F00FF 3300FF 3F00FF 3300FF 3F00FF 3300FF 3200FF 3F00FF 3300FF 3F00FF 3300FF 3200FF 3F00FF 3300FF 3F00FF 3300FF 3200FF 3F00FF 3300FF 3F00FF 3300FF
SQLDHROW	077800050302E0	0671E0 D00001

DRDA Type	MDD Descriptor in HEX	SDA, GDA, CPT, or RLO Descriptor in HEX
SQLDCROW	077800050302E5	0671E5 D50001
SQLCNROW	077800050302E6	0671E6 D60001
SQLTOKROW	077800050302E7	0671E7 D70001
SQLDIAGCI	077800050402F5	0971F5 680001 E50000
SQLDIAGCN	077800050402F6	0971F6 680001 E60000
SQLDCTOKS	077800050402F7	0971F7 680001 E70000

### 5.8.1.3 Late Data Unit Descriptors

The Late Data Unit Descriptors in [Table 5-22](#) apply for both SQLAM Level 3 and SQLAM Level 4. The only exception is SQLDTAMRW, which is not supported at SQLAM Level 3.

**Table 5-22** 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 5-23](#) (on page 401), which is resident in a QTDSQL370 machine and is called STATS.

**Table 5-23** 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 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 5-24 (on page 402). This table does not show the DDM command proper and its parameters.

**Table 5-24** EXECUTE IMMEDIATE Command Data

Reference	HEX Representation	Description
SQLSTT	00242414	DDM Length and Code Point for SQL Statement
SQLSTT	FF	NOCM — Null
SQLSTT	00 0000001E 4752414E542053454C454354204F4E205354 41545320544F204252554345	NOCS — Length and "GRANT SELECT ON STATS TO BRUCE"

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 5-25 (on page 402).)

**Table 5-25** EXECUTE IMMEDIATE Reply Data

Reference	HEX Representation	Description
SQLCARD	00052408	DDM codepoint 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 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 5-26 shows the actual bytes that flow to show the command data. It does not show the DDM command proper and its parameters.

**Table 5-26** Open Query Command Data

Reference	HEX Representation	Description
OBJDSS	0027D003 xxxx	Object Data Stream Structure
SQLDTA	00212412	DDM codepoint for SQL objects with FD:OCA Descriptors and Data
FDODSC	00100010	DDM codepoint for FD:OCA Descriptor objects  <b>Note:</b> 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 — 8-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 codepoint 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 the Passing Warnings to the Application Requester (WN Rules) in Section 7.20 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 5-27 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	001C241A	DDM codepoint for FD:OCA Descriptor objects <b>Note:</b> MDD/SDA pairs for unusual data would be here if they were required.
SQLDTAGRP	0976D0	Start Nullable Group Descriptor — GDA Header
SQLDTAGRP	320014 CB8009	Continue — VARCHAR(20) nullable CLOB(2M)
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	0458241B	DDM codepoint for FD:OCA Data objects
QRYDTA	FF00 0003 C2D6C2 00 01 00000000 00000400 C2CFC2...	1st Row — Null SQLCA, non-null row BOB(3) not null, mode X'01', length 1K, "BOB's Resume..."
QRYDTA	FF00 0003 D1C9D4 FF	2nd Row — Null SQLCA, non-null row JIM(3) null resume
QRYDTA	FF00 0003 E2C1D4 00 02 00000000 00019000	3rd Row — Null SQLCA, non-null row SAM(3) not null, mode X'02', length 100K
QRYDTA	FF00 0003 D1D6C5 00 01 00000000 00000000	4th Row — Null SQLCA, non-null row JOE(3) not null, mode X'01', length 0 (0-length resume)
QRYDTA	FF00 0003 D9D6C4 00 03 00000000 00100001 pppppppp pppppppp	5th Row — Null SQLCA, non-null row ROD(3) not null, mode X'03', length 1M+1, Progressive Reference (p..p)
OBJDSS — Continued	FFFFD053xxxx	Object Data Stream Structure
EXTDTA	800C 146C 00000000 00019001 00 E2C1D4...	DDM codepoint for FD:OCA Externalized Data length 100K, not-null, "SAM's Resume..." (32748 bytes of data)
OBJDSS — Continued	FFFF ...	2nd part of "SAM's Resume..." (32765 bytes of data)
OBJDSS — Continued	FFFF ...	3rd part of "SAM's Resume..." (32765 bytes of data)

Reference	HEX Representation	Description
OBJDSS — Continued	101C ...	4th (last) part of "SAM's Resume..." (4122 bytes of data)

The EOF SQLCA becomes null after SQLERRPROC because of the presence of the nullable group SQLCAXGRP inside the SQLCARD.

The entire answer set was short enough to be included in the first query block returned as a result of the command. The program has issued 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, the server has to decide whether or not the query should be closed implicitly based on the type of query, and the *qryclsimp* parameter as previously sent on the OPNQRY command. In this example, the server indicates that it has closed the query implicitly by sending back an ENDQRYRM reply message and its associated SQLCARD reply data object that contains SQLSTATE 02000. 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 00000 SQLCA to the Close Cursor request when the application issues it.

## 5.8.2.3 Input Variable Arrays SQL Request

An example of an SQL statement with input variable arrays:

```
EXEC SQL INSERT INTO STATS :NBR ROWS VALUES (:HA1, :HA2, :HA3)
```

Variable NBR is declared as a one-byte integer and has the value 2. The variable arrays are declared as arrays with 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 requester to the server using the EXCSQLSTT command.

Table 5-28 shows the actual bytes that flow to show the SQLDTA command data. It does not show the proper command and its parameters.

**Table 5-28** Input Variable Array Command Data

Reference	HEX Representation	Description
OBJDSS	006F D003 0001	Object Data Stream Structure
SQLDTA	0069 2412	Input Data Object
FDOEXT	0014 147B	Extent Object
SQLARYGRP	0000 0002	Extent for :HA1
SQLARYGRP	0000 0002	Extent for :HA2
SQLARYGRP	0000 0002	Extent for :HA3
SQLARYGRP	0000 0001	Extent for :NBR
FDODSC	0019 0010	Input Data Descriptor
SQLDTAGRP	0F76D0	Nullable GDA Header
SQLDTAGRP	050002	:HA1 SMALLINT Array Descriptor
SQLDTAGRP	050002	:HA2 SMALLINT Array Descriptor
SQLDTAGRP	320014	:HA3 VARCHAR Array Descriptor
SQLDTAGRP	020004	:NBR INTEGER Array Descriptor
SQLDTA	0671E4 D00001	Row Descriptor
FDODTA	0024 147A	Input Data
FDODTA	00	Null Indicator for Variable Array Group
FDODTA	00 0028 00 0019	:HA1 Nullable SMALLINT[2] Data Array Values
FDODTA	00 00AA 00 00A0	:HA2 Nullable SMALLINT[2] Data Array Values
FDODTA	0006D9D6C2C5D9E2 0005E2E3C5E5C5	:HA3 VARCHAR[20] Data Array Values
FDODTA	0000 0002	:NBR INTEGER[1] Data Array Value
FDOOFF	0014 147D	Offset Object
SQLARYGRP	0000 0000	Offset for :HA1
SQLARYGRP	0000 0006	Offset for :HA2
SQLARYGRP	0000 000C	Offset for :HA3
SQLARYGRP	0000 001B	Offset for :NBR

## 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 5-29 shows the data that flows in the OBJDSS (object data stream structure) which follows the EXCSQLSTT command. Notice only host variable parameters flow.

Table 5-30 shows an example reply data stream. Notice the use of -128 (X'80') indicator values in the FDOTA to flag the second and third parameters as being INPUT only.

**Table 5-29** 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 codepoint (LLCP) for SQLDTA
FDODSC	0016 0010	DDM Length and codepoint (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 codepoint (LLCP) for FDOTA
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 5-30** 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 codepoint (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 4-28 (on page 178). 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 5-31** 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 codepoint (LLCP) for RDB Result Set Reply Message
SVRCOD	00061149 0000	RDB Result Set Reply Message Severity Code
PKGSNLST	008C2139	DDM length and codepoint (LLCP) for RDB Package Name, Consistency Token, and Section Number List
PKGNAMECSN	00442113 xxxxxxxx xxxxxxx xxxxxxxx xxxxxxx xxxxxxxx xxxxxxx xxxxxxxx xxxxxxx xxxxxxxx xxxxxxx xxxxxxxx xxxxxxx xxxxxxxx xxxxxxx xxxxxxxx xxxxxxx	PKGNAMECSN for result set #1
PKGNAMECSN	00442113 xxxxxxxx xxxxxxx xxxxxxxx xxxxxxx xxxxxxxx xxxxxxx xxxxxxxx xxxxxxx xxxxxxxx xxxxxxx xxxxxxxx xxxxxxx xxxxxxxx xxxxxxx xxxxxxxx xxxxxxx	PKGNAMECSN for result set #2
OBJDSS	0065D003 xxxx	Object Data Stream Structure
SQLCARD	00052408	DDM length and codepoint (LLCP) for SQL Communication Area Reply Data
SQLCARD	FF	Null SQLCARD
SQLRSLRD	005A240E	DDM length and codepoint (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

## 5.8.2.6 Enabling Dynamic Data Format

For this example, assume that STATS table has an additional nullable CLOB(2M) column called RESUME having the following content:

**Table 5-32** STATS Sample Table (Expanded)

<b>NAME Varchar(20) Not Null</b>	<b>RESUME CLOB(2M) Nullable</b>
BOB	"BOB's Resume...." length of 1K bytes
JIM	null
SAM	"SAM's Resume...", length of 100K bytes
JOE	"", length of 0 byte
ROD	"ROD's Resume...", length of 1M + 1 bytes

These are the SQL statements for this example (in PL/I):

```
EXEC SQL DECLARE mycursor CURSOR FOR SELECT NAME, RESUME FROM STATS;
EXEC SQL OPEN mycursor;
```

Assume that the following instance variables are set with OPNQRY:

```
DYNDTAFMT=TRUE
SMLDTASZ=32767 (same as default)
MEDDTASZ=1048576 (same as default 1048576)
OUTOVROPT=OUTOVRNON
```

Table 5-33 Open Query Reply Data

Reference	HEX Representation	Description
RPYDSS	0016D052xxxx	Reply Data Stream Structure
OPNQRYRM	00102205 000611490000 000621022417	Open Query Reply Message Severity Code Value Limited Block Protocol
OBJDSS	047AD053xxxx	Object Data Stream Structure
QRYDSC	001C241A	DDM codepoint for FD:OCA Descriptor objects Note: MDD/SDA pairs for unusual data would be here if they were required.
SQLDTAGRP	0976D0	Start Nullable Group Descriptor — GDA Header
SQLDTAGRP	320014 CB8009	Continue — VARCHAR(20) nullable CLOB(2M)
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	0458241B	DDM Codepoint for FD:OCA Data objects
QRYDTA	FF00 0003 C2D6C2 !00 01 00000000 00000400 C2CFC2...	1st Row — Null SQLCA, non-null row BOB(3) not null, mode X'01', length 1K, "BOB's Resume..."
QRYDTA	FF00 0003 D1C9D4 FF	2nd Row — Null SQLCA, non-null row JIM(3) null resume
QRYDTA	FF00 0003 E2C1D4 00 02 00000000 00019000	3rd Row — Null SQLCA, non-null row SAM(3) not null, mode X'02', length 100K
QRYDTA	FF00 0003 D1D6C5 00 01 00000000 00000000	4th Row — Null SQLCA, non-null row JOE(3) not null, mode X'01', length 0 (0-length resume)
QRYDTA	FF00 0003 D9D6C4 00 03 00000000 00100001 pppppppp pppppppp	5th Row — Null SQLCA, non-null row ROD(3) not null, mode X'03', length 1M+1, Progressive Reference (p..p)
OBJDSS (continued)	FFFFD053xxxx	Object Data Stream Structure
EXTDTA	800C 146C 00000000 00019001 00 E2C1D4...	DDM codepoint for FD:OCA Externalized Data length 100K, not-null, "SAM's Resume..." (32748 bytes of data)
OBJDSS (continued)	FFFF ...	2nd part of "SAM's Resume..." (32765 bytes of data)
OBJDSS (continued)	FFFF ...	3rd part of "SAM's Resume..." (32765 bytes of data)

Reference	HEX Representation	Description
OBJDSS (continued)	101C ...	4th (last) part of "SAM's Resume..." (4122 bytes of data)

Since limited block protocol was used, RTNEXTDTA=RTNEXTALL was applied to be used when Dynamic Data Format is enabled. Although the answer set contained a LOB column, OUTOVRPT was OUTOVRNON along with limited block protocol and RTNEXTDTA being RTNEXTALL made it possible to return QRYDTA and all of its associated EXTDTAs in response to OPNQRY. However, although the entire answer set was short enough to be included in the first query block returned as a result of the command, no implicit close was done because a progressive reference had just been returned to the application requester. The application requester might subsequently issue a GETNXTCHK command with the progressive reference to retrieve "ROD's Resume..." a chunk at a time.



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** 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
    255      (255 bytes total)
```

The character string that represents an end user-name within the DRDA flows has a maximum length of 255 bytes and must consist of characters in the character set identified by the CCSIDMGR for the connection. Restrictions on the size and character set may be imposed by the local security server on the target server or the network protocols<sup>48</sup> used to connect to the RDB.

#### 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.

---

48. The SNA LU6.2 ALLOCATE verb restricts the USER\_ID to a maximum length of 8 bytes and must consist of letters (A through Z) and numerics (0 through 9).

## 6.2 RDBs

The name for a relational database is RDB\_NAME.

### Syntax

RDB\_NAME  
255 (255 bytes total, the first 6 bytes are registered)

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 255 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. A relational database can be unpartitioned, in which case all user table data resides at one location. If, instead, data in user tables is spread across multiple locations, the relational database is considered partitioned.

**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  
  255 . 255 . 255 (765 bytes plus period delimiters)
```

[Section 6.2](#) defines the syntax of RDB\_NAME.COLLECTION and OBJECTID which 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 255 bytes. The maximum length of OBJECTID is 255 bytes.

### Semantics

#### RDB\_NAME

Identifies the relational database whose catalog contains information for the object. Refer to [Section 6.2](#) 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
  255      255      255  (767 bytes total)
```

[Section 6.2](#) 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 255 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](#) 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 419)) 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 Command Source Identifiers

A single database connection can be used for multiplexing requests from different application sources. The sources can be nested UDFs or stored procedures all stemming from a single user application, or they may be distinct applications altogether. In this environment, requests from multiple application sources may try to operate against an identical section within a package. For example, multiple sources may attempt to prepare different SQL statements into a single section. In order to avoid collisions amongst such requests, the application requester has the option of flowing a command source identifier (CMDSRCID) to the server for each request. The command source identifier complements the package name, consistency token, and section number (PKGNAMECSN) by uniquely identifying the source of the database request, thereby distinguishing it from an otherwise identical request, albeit from a different application source. The command source identifier can be specified on the following commands: CLSQRY, CNTQRY, DSCSQLSTT, EXCSQLIMM, EXCSQLSTT, OPNQRY, and PRPSQLSTT.

For cursor operations, the command source identifier, in conjunction with the package name, consistency token, and section number, uniquely identifies a query for a particular application source. Furthermore, it is possible to have multiple instances of such a query through the use of the query instance identifier (QRYINSID). For more information on the query instances, refer to [Section 4.4.6](#) (on page 145).

## 6.4.5 Package Collection Resolution

Two mechanisms currently exist for managing package collection resolution (also known in some environments as *schema resolution*) at the requester. The first mechanism is the specification of a default collection ID inside the PKGNAMECSN that is specified at execution time. The second mechanism is the use of the SET CURRENT PACKAGESET command to indicate what the qualifier is for a package that is to be invoked for all subsequent SQL operations. The CURRENT PACKAGESET value, if set, takes precedence over the default collection ID.

In order to support package switching functionality for SQL applications, it is desirable to be able to select the appropriate package from a list of package collections. Given an ordered list of package qualifiers, the server will select the first collection in which a match is found for the package name, consistency token, and version ID. The CURRENT PACKAGE PATH value provides such an ordered list to allow for server management of package collection resolution, and this value will override the value provided in the PKGNAMECSN parameter flowed at execution time. In other words, at the server, the CURRENT PACKAGE PATH value (if set) will override any package collection resolution ordering defined by either the collection inside the PKGNAMECSN parameter or a collection explicitly indicated via the SET CURRENT

PACKAGESET command. The CURRENT PACKAGE PATH value does not override the collection specified as part of the PKGNAM or PKGNAMCT parameters.

## 6.5 Stored Procedure Names

The qualified form for a stored procedure name is RDB\_NAME.COLLECTION.PROCEDURE.

### Syntax

```
RDB_NAME . COLLECTION . PROCEDURE
  255   .   255       .   255   (765 bytes plus period delimiters)
```

[Section 6.2](#) defines the syntax of RDB\_NAME.COLLECTION and PROCEDURE which 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 255 bytes. The maximum length of PROCEDURE is 255 bytes.

### Semantics

#### RDB\_NAME

Identifies the relational database whose catalog contains information for the procedure. Refer to [Section 6.2](#) 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](#) 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](#)
- [Section 13.6.3](#)



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 247). [Chapter 5](#) precisely describes the description and formats of data exchanged between application requesters and application servers. See [Section 5.3](#) for rules pertaining to this topic.

The following sections define the DRDA rules between an application requester and an application server. The rules are equivalent between an application server and a database server 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 Atomic Chaining (AC Rules)

**AC1** Only EXCSQLSTT commands can be part of an atomic chain. The chain is considered atomic if in the event that any one of the EXCSQLSTT commands fails, then all other changes made to the database under this chain will be undone. The optional *atmind* parameter, if flown on an EXCSQLSTT command within the atomic chain, must be set to its default value of X'00'. Otherwise, the application server must return SYNTAXRM with *synerrcd* set to X'1E'.

There can be zero or more EXCSQLSTT commands within the atomic chain.

**AC2** The atomic chain is initiated by the BGNATMCHN command and terminated by the ENDATMCHN command.

If the chain as initiated by the BGNATMCHN command contains any command other than EXCSQLSTT and ENDATMCHN commands, the application server must return PRCCNVRM with *prccnvc* set to X'1C'.

If the ENDATMCHN command is sent without a prior matching BGNATMCHN command, the application server must return PRCCNVRM with *prccnvc* set to X'1D'.

**AC3** Processing of an atomic chain by the application server terminates when an error occurs on an EXCSQLSTT command in the chain, or when the ENDATMCHN command has been processed, whichever occurs first.

Each EXCSQLSTT command that gets processed by the application server in the atomic chain except for the one returning an error that terminates processing of the chain must result in an SQLCARD or SQLDTARD reply data object indicating success.

The last EXCSQLSTT command that gets processed by the application server is either also the last EXCSQLSTT command in the chain, or is one that results in an error. If an EXCSQLSTT command results in an error, then the reply to this EXCSQLSTT must be an error reply message indicating the termination of processing of the atomic chain, to

be optionally followed by an SQLCARD. If this is an SQL error which does not otherwise have an error reply message associated with it, the application server must send back an SQLERRRM reply message to be followed by the SQLCARD in accordance with Rule CU4 in [Section 7.6](#) (on page 436).

- AC4** If processing of an atomic chain terminates because of an error condition or a valid INTRDBRQS command, all changes made as a result of previous successful statement executions earlier in the atomic chain are undone.

## 7.2 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](#)
- [Section 13.6.2.1](#)

**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.

A connection between an application requester and an application server using distributed unit of work can be either protected by the SyncPoint manager or the XA manager. If either the application requester or application server does not support a protected connection, the connection should be established as a DUOW unprotected connection.

See rules usage for environment in these sections:

- [Section 12.8.2.1](#)
- [Section 13.6.2.1](#)

**CA5** ACCRDB or INTRDBRQS 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](#)

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 CCSIDDBC, CCSIDMBC, and CCSIDXML 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.

An optional SQLCARD reply data object may follow the ACCRDBRM reply message for a successful connection if the receiver of ACCRDB encounters an SQL warning condition upon a successful connection and/or if there are server-specific connect tokens that need to be returned.

- 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 CCSIDDBC, CCSIDMBC, and CCSIDXML 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.

If the ACCRDBRM reply message is followed by an optional SQLCARD reply data object, the SQL code contained therein must be greater than or equal to zero to indicate the connection is successful. If the SQL code contained in this SQLCARD is less than zero, the application requester must report the error to the application with an SQLSTATE of X'58009'. Otherwise, how the application requester processes this SQLCARD is not defined by DRDA. For example, the application may choose to return the contents of this SQLCARD directly to the application through the SQLCA. Or if there is an aforementioned local CCSIDMBC/CCSIDDBC warning, or a server CCSIDMBC/CCSIDDBC warning in the ACCRDBRM as documented under Rule CA10, the requester may choose to merge the contents of this SQLCARD and the CCSID warning into one SQLCA to return to the application. The application requester may also ignore this SQLCARD altogether.

- 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).

### 7.3 Mapping of Reply Messages to SQLSTATEs (CD Rules)

- CD1 If an application requester receives a valid reply message with a valid *svrcod*, the application requester must return the SQLSTATE listed in [Section 4.3.1](#) (on page 70). 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 SQLSTATE 58018.
- CD2 If an SQLCARD accompanies a reply message, the SQLCODE and SQLSTATE in the SQLCARD should be passed to the application.

### 7.4 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.

## 7.5 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 Rule CR6 for the SQLSTATES to return.

See rules usage for environment in these sections:

- [Section 12.8.2.2](#)
- [Section 13.6.2.2](#)

**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.

The above implies only to connections that are not using the services of the XA manager (see footnote †). For XAMGR protected connections, the application must initiate commit or rollback before terminating. If the application does not explicitly drive a two-phase commit or roll back on normal termination, or terminates abnormally, the application server will implement the presumed abort protocol and implicitly roll back the transaction in both cases.

**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](#)

Not applicable to TCP/IP.

- CR9** An application server using distributed unit of work not protected by the sync point manager or XA manager (see footnote †) 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 or using the XA manager (see footnote †) 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 not protected by the sync point manager or XA manager (see footnote †) that is allowed to commit via the *rdbcmtok* parameter.

- CR12** An application server using distributed unit of work or the XA manager (see footnote †) 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 or XA manager (see footnote †), 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 or XA manager (see footnote †) 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.

- CR15** An EXCSQLSTT command in an atomic chain enclosed within a BGNATMCHN-ENDATMCHN command pair must not invoke a commit or rollback either implicitly or explicitly through a commit or rollback request. If the application server receives such a command, it should terminate processing of the atomic chain with SQLSTATE '2D522'.
- CR16** An interrupt request (INTRDBRQS) has no effect on the execution of a commit or rollback.
- CR17** When using the services of the XAMGR, the application must protect all connections. All XAMGR protected connections must comply with the following. The application server must reject any request that does not comply.
- All Global Transactions must be registered before any work can be performed on that connection. Any work started without registering will be considered a Local Transaction. Attempts to mix a Global Transaction with a Local Transaction or *vice versa* should be rejected.
  - The application must signal the end of all Global Transactions, but should not do so for Local Transactions. Attempts to do so should be rejected by the application server.
  - The application must use the two-phase protocols to commit/rollback all Global Transactions. For Local Transactions, the application must drive a local commit/rollback using the RDB manager (that is, RDBCMM/RDBRLLBCK).
  - A connection is associated with an XID when it has successfully registered with the application server. The XID is dissociated from the connection at end time, unless the application server indicates to the application requester that

connection should not be dissociated from the transaction.

- The application may prepare, commit, or roll back any XID on any XAMGR protected connection, as long as that XID has completed, and does not exist in a suspended state on any connection.

**CR18** Connections protected by the sync point manager or XA manager, where the connection is active with a Global Transaction, cannot issue any static/dynamic commit or rollback. Any attempts by the application or stored procedure to perform any sort of Static/Dynamic commit or rollback should be rejected. If the application is using the XA manager, but the connection is active with a Local Transaction, then any static/dynamic commit or rollback is valid from the application or stored procedure only when the RDBCMTOK is set to TRUE). Following are the SQLSTATEs that should be relayed back to the application when either a static/dynamic commit or rollback is rejected.

- SQLSTATE 2D521 for both static COMMIT and ROLLBACK
- SQLSTATE 2D528 for both dynamic COMMIT and ROLLBACK

**CR19** If, over a CMNSYNCPT or SYNCPTMGR-protected connection, the application server indicates via the *unpupd* parameter of an RDBUPDRM reply message that an unprotected update has been performed downstream as per the *unpupdalw* parameter, as specified previously on the ACCRDB command, the application requester must for this unit of work either ensure that all other database connections are read-only, or issue a rollback.

In addition, if an unprotected update has been performed downstream in violation of the *unpupdalw* parameter as specified on the previous ACCRDB command, the application requester must roll back the unit of work.

---

† These rules only apply to XAMGR protected connections that are in a Global Transaction, and do not apply to connections in a Local Transaction. See Rule CR16 for more details.

## 7.6 Connection Usage (CU Rules)

See the DDM Reference for descriptions of the DDM commands.

- CU2** The first DDM command required to flow over a connection is the EXCSAT command. The EXCSAT command can flow anytime on a connection.
- CU3** The first DDM command is either ACCRDB or INTRDBRQS. SQLAM uses DDM commands in the QDDRDBD dictionary. This dictionary contains all DDM classes that describe the commands, replies, and data objects required to communicate with an RDB.
- CU4** If the application server desires to terminate DDM command chaining, and there is no appropriate DDM reply message 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.
- CU7** On the same connection, the INTRDBRQS is not valid after an ACCRDB command.
- CU8** On the same connection, the ACCRDB is not valid after an INTRDBRQS command.
- 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'.
- CU14** A network connection can only be reused for another application at the end of a transaction.
- CU15** The fully qualified package name and package consistency token are not required to be specified on every SQL-related request. If the package name and consistency token are not specified on a request, the last request that specified the package name and consistency token is used to identify the package name and consistency token for the request. The package section number is not optional and is required if the package name and consistency token are not specified. If the package name and consistency token were not specified on a previous request to establish the default, the request is rejected by the application server with a conversational protocol error with the error code set to X'20' (default package name not established).

- CU16** A network connection established as a remote unit of work connection cannot be reused by the requester for another application as a distributed unit of work connection. A network connection established as a distributed unit of work connection cannot be reused by the requester as a remote unit of work connection.
- CU17** A network connection can only be reused for another application on a connection boundary if and only if the application releases or disconnects from the connection (such as issuing an SQL RELEASE or when the application terminates). When a connection is reused, the requester issues the same commands used to establish the connection and access the RDB (that is, EXCSAT, ACCSEC, SECCHK, ACCRDB). The server must close or destroy all RDB resources associated with the current application prior to executing any new commands for the new application. All held cursors are closed. All temp tables are destroyed. Special registers are set to default values. The application execution environment is set to a default state as if it was a new connection being established.
- CU18** A network connection can only be reused for another application on a transaction boundary if and only if the application server returns an indicator on the availability of the connection to be reused by another application. The following illustrates which DDM requests represent a transactional boundary:

Connection Type	Transactional Boundary (DDM Request)
RUOW - Remote Unit of Work	RDBCMM - Commit RDBRLLBCK - Rollback
SYNCPTMGR protected connections	SYNCCTL(Commit) SYNCCTL(Rollback)
XAMGR protected connections in a Global Transaction	SYNCCTL(End) with flag TMSUCCESS, TMFAIL, or TMSUSPEND (if application server can support dissociation of transaction)
XAMGR protected connections in a Local Transaction	RDBCMM - Commit RDBRLLBCK - Rollback

At the transactional boundary (for all connection types), the target server can only indicate that the connection is available for reuse when the following conditions are met:

1. The transaction closed all cursors related to the application.
2. The transaction closed all RDB resources associated with the application (for example, temp tables).
3. The transaction did not establish a special execution environment that may impact the execution of a transaction by another application (for example, the degree special register changed to a non-default value).
4. The requester requested the connection be released for reuse.
5. The server indicated that the connection can be released for reuse.

For XAMGR protected connection, that performed a SYNCCTL(End) with TMSUSPEND. If the application requester cannot satisfy conditions 1 to 3, and resume them later, then it should *not* send the reuse indicator. And both application requester and application server should remain associated with the XID. If the application server cannot satisfy conditions 1 to 3 and resume them later, then it should respond with RLSCON(NO). Both application requester and application server should remain

associated with the XID.

If both application requester and application server can save conditions 1 to 3, and resume them later, it is expected that both should dissociate the connection from the XID, and satisfy conditions 4 and 5; that is, the application requester should send the reuse indicator, and the application server should respond with REUSE.

- CU19** If a server returns a group of SET statements to be used to establish the application execution environment for the next transaction, these SET statements must be sent to the server using the set environment (EXCSQLSTT) command prior to executing another transaction for the application. If a server is an intermediate database server, all SET statements sent to all database servers involved in the transaction must be included in the group of SET statements. When a released for reuse connection is reused, the requester must issue the same commands used to establish the connection, access the RDB, and then issue the set execution environment command with any SET statements returned on the last commit or rollback executed on behalf of the application (that is, EXCSAT, ACCSEC, SECCHK, ACCRDB, EXCSQLSET).
- CU20** Prior to a released network connection being reused, the server must close or destroy all RDB resources associated with the current application. All held cursors are closed. All temp tables are destroyed. Special registers are set to default values. The application execution environment is set to a default state as if it was a new connection being established.
- CU21** If a requester reuses a network connection without the server indicating the connection is available for reuse, all resources associated with the connection are released as per Rule CU19.
- CU22** SNA security cannot be used to authenticate users on a reused network connection.
- CU23** If a reply data object to be sent does not conform to the manager level selected for use between the server and the requester as described in Rule CU10, then the server adds a MGRLVLOVR object to reply data objects, as necessary, to correctly describe the format and content of the object being sent.
- For the reply objects returned for a given command, the MGRLVLOVR object is sent as a reply data object. The format and content of all the following reply data objects of that command are affected.
- CU24** MGRLVLOVR may be specified as many times as necessary to correctly describe all objects returned in reply to a command. Each MGRLVLOVR remains in effect for all subsequent data objects until the end of the reply, or until another MGRLVLOVR is encountered with its own override, whichever occurs first.
- The override is in effect for only the reply to one command, except as specified by Rules CU27 and CU28.
- CU25** The manager level specified in the MGRLVLOVR parameter must be a value less than or equal to the corresponding value selected for use between the requester and the server as described in Rule CU10. A PRCCNVRM with PRCCNVCD of '22'X (an override manager level in an MGRLVLOVR is greater than the corresponding manager level negotiated between the requester and server by EXCSAT/EXCSATRD) is returned if this rule is violated.
- CU26** The only manager whose level that may be overridden by use of the MGRLVLOVR parameter is the SQL Application Manager.

- CU27** If a requester cannot process a new value for MGRLVLOVR (assuming that the value is valid according to the other CU Rules) that is received from a server as part of a reply data object, then it must produce an SQLCA for the application. The SQLCA indicates SQLSTATE 56072, specifying the parameter that the server requested, but that the requester cannot support.
- CU28** The manager level that applies to all data from a single query received in QRYDTA objects (SQLCAs and user data) and their associated EXTDTA objects is determined by the default manager level selected for use between the requester and the server as described in Rule CU10 or by the override value in effect at the time the QRYDSC is received.
- CU29** The Query Processing rules in effect for the reply objects returned for a query or result set are determined by the default manager level selected for use between the requester and the server as described in Rule CU10 or by the override value in effect at the time the QRYDSC is received, except as follows:

In response to OPNQRY or EXCSQLSTT for a stored procedure result set:

- If the first QRYDSC object is preceded by a MGRLVLOVR object with an SQLAM level of 6 or lower, then the OPNQRYRM is not to contain a QRYBLKTYP parameter.
- If the first QRYDSC object is preceded by a MGRLVLOVR object with an SQLAM level of 6 or lower, then one or more QRYDSC objects will follow in sequence after the MGRLVLOVR object. The QRYDSC objects are not to be considered as part of a query block. The QRYDSC or QRYDSC objects will be formatted according to the Blocking rules in effect for the override SQLAM level, except that the first QRYDSC and the last QRYDSC object may be smaller than the specified QRYBLKSZ.
- If the first QRYDSC object is preceded by a MGRLVLOVR object with an SQLAM level of 6 or lower, then one or more optional SQLCINRD objects will follow in sequence after the last QRYDSC object. The optional SQLCINRD objects are not to be considered as part of a query block. The SQLCINRD object or SQLCINRD objects will be formatted according to the Blocking rules in effect for the override SQLAM level, except that the first SQLCINRD and the last SQLCINRD object may be smaller than the specified QRYBLKSZ.
- If the first QRYDSC object is preceded by a MGRLVLOVR object with an SQLAM level of 6 or lower, then one or more QRYDTA objects will follow in sequence after the last optional SQLCINRD object, if any optional SQLCINRD objects are present, or after the last QRYDSC, if no optional SQLCINRD objects are present. The QRYDTA objects are not to be considered as part of a query block. The QRYDTA or QRYDTA objects will be formatted according to the Blocking rules in effect for the override SQLAM level, except that the first QRYDTA and the last QRYDTA object may be smaller than the specified QRYBLKSZ. If only one QRYDTA is returned, it contains only a partial row.
- If the first QRYDSC object is preceded by a MGRLVLOVR object with an SQLAM level of 6 or lower, then an ENDQRYRM and SQLCARD object may be returned for the query for result set. The ENDQRYRM and SQLCARD objects are not to be considered as part of a query block.

These exceptions do not apply to the responses to a CNTQRY request.

## 7.7 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 basic floating point numbers (BF16, BF8, BF4), use the default rounding rule given in Rule DT7 (see [Section 7.9](#) (on page 448)).

When converting decimal floating point numbers (DFP) to a basic floating point (BF16, BF8, BF4) format, the rounding rule is implementation-dependent. The default rounding rule is to round to the nearest value, but in the case of two nearest values, round so that the final digit is even. If the application requester provides an implementation-specific rounding rule, that rounding rule applies instead.

Exceptions may occur when converting from DRDA data stream data types to application variable data types. The application program receives an SQLSTATE of X'22003' 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
XML string internal encoding	BLOB <sup>49</sup>	8
XML string external encoding	CLOB or DBCLOB	8
Fixed binary	char(n) for BIT <sup>50</sup>	8
Variable binary	varchar(n) for BIT <sup>51</sup>	8
Decimal floating point <sup>52</sup>	8-byte basic floating point of the type corresponding to the TYPDEFNAM of the requester	8
Boolean	2-byte integer	8

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.

If the requester performs the defined conversion for a data value, but the conversion results in an error, the requester fails the associated SQL request and returns SQLSTATE 22003.

**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:

- 
49. XML String Internal Encoding can be converted to CLOB or DBCLOB if the sender can ensure the external encoding will not corrupt the internal encoding (e.g., no internal encoding tag but assumed UTF-8 or UTF-16 can be returned as CLOB(1208) or DCLOB(1200)).
  50. Although DRDA defines these as compatible mappings, these data types cannot be used interchangeably. The application is responsible for managing the differences in SQL semantics between character and binary data. These include, for example, differences in padding bytes and differences in comparison operations.
  51. Although DRDA defines these as compatible mappings, these data types cannot be used interchangeably. The application is responsible for managing the differences in SQL semantics between character and binary data. These include, for example, differences in padding bytes and differences in comparison operations.
  52. If a decimal float is downgraded to a hexadecimal or binary float, there may be a loss of precision of which the receiver is not aware. DRDA supports downgrading the data type to facilitate retrieval of data; for example, by middleware products that routinely perform SELECT \* operations against the server. It is, however, the application's responsibility to be aware of this possibility.

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
XML string internal encoding	BLOB <sup>53</sup>	8
XML string external encoding	CLOB or DBCLOB	8
Fixed binary	char(n) for BIT <sup>54</sup>	8
Variable binary	varchar(n) for BIT <sup>55</sup>	8
Decimal floating point <sup>56</sup>	8-byte basic floating point of the type corresponding to the TYPDEFNAM of the server	8
Boolean	2-byte integer	8

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.

If the server performs the defined conversion for a data value, but the conversion results in an error, the server fails the associated SQL request and returns SQLSTATE 22003.

---

53. XML String Internal Encoding can be converted to CLOB or DBCLOB if the sender can ensure the external encoding will not corrupt the internal encoding (e.g., no internal encoding tag but assumed UTF-8 or UTF-16 can be returned as CLOB(1208) or DCLOB(1200)).

54. Although DRDA defines these as compatible mappings, these data types cannot be used interchangeably. The application is responsible for managing the differences in SQL semantics between character and binary data. These include, for example, differences in padding bytes and differences in comparison operations.

55. Although DRDA defines these as compatible mappings, these data types cannot be used interchangeably. The application is responsible for managing the differences in SQL semantics between character and binary data. These include, for example, differences in padding bytes and differences in comparison operations.

56. If a decimal float is downgraded to a hexadecimal or binary float, there may be a loss of precision of which the receiver is not aware. DRDA supports downgrading the data type to facilitate retrieval of data; for example, by middleware products that routinely perform SELECT \* operations against the server. It is, however, the application's responsibility to be aware of this possibility.

- 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.

## 7.8 Data Format (DF Rules)

Data Format (DF Rules) specify how an FD:OCA Generalized String is formatted in an QRYDTA and optionally in an EXTDTA. If the FD:OCA Generalized String is a column in an answer set or result set, it is assumed that no OUTOVR object is specified or there is no override triplet for the column in the OUTOVR object. If there is an OUTOVR object with an override triplet for the column in the answer set or result set, see .cX `override_output` for more details.

Dynamic Data Format allows FD:OCA Generalized String data in an answer set to be returned in a representation that is determined by the application server at the time when the data is retrieved based on its actual length. This provides the application server with the ability not to flow such data as an EXTDTA object when it is inefficient or impractical to do so. Each representation is called a *mode*, and the following modes are supported:

- X'01' - data in QRYDTA

QRYDTA			
optional null-ind	mode 0x01	length	data
1 byte	1 byte	8 bytes	<i>n</i> bytes

- X'02' - data in EXTDTA

QRYDTA			EXTDTA	
optional null-ind	mode 0x02	length	optional null-ind	data
1 byte	1 byte	8 bytes	1 byte	<i>n</i> bytes

- X'03' - data as a progressive reference

QRYDTA			
optional null-ind	mode 0x03	length	progressive reference
1 byte	1 byte	8 bytes	8 bytes

**DF1** Dynamic Data Format is enabled by the application requester by specifying `DYNDTAFMT=TRUE` on the `OPNQRY` or `EXCSQLSTT` command that invokes a stored procedure that returns one or more result sets. If `DYNDTAFMT` is set to `TRUE`, then `SMLDTASZ` and `MEDDTASZ` may also be specified. Dynamic Data Format is not enabled when `DYNDTAFMT` is not specified or if `DYNDTAFMT` is set to `FALSE`.

**DF2** `SMLDTASZ` specifies the maximum size of the data in bytes to be flown in Mode X'01' of Dynamic Data Format. The range of valid values is from 0 to `QRYBLKSZ`. If the value is outside of the valid range or it is greater than `MEDDTASZ`, the server rejects it with a `PRCCNVRM`. Data with length greater than `SMLDTASZ` will be sent in Mode X'02' or Mode X'03'. A value of 0 means that no data is to be sent in Mode X'01'. If `SMLDTASZ` is not specified, the server assumes a default value of 32767. `SMLDTASZ` is ignorable unless `DYNDTAFMT` is also set to `TRUE`.

**DF3** `MEDDTASZ` specifies the maximum size of the data in bytes to be flown in Mode X'02' of Dynamic Data Format. The range of valid values is from `SMLDTASZ` to 2147483647. If the value is outside of the valid range or it is less than `SMLDTASZ`, the server rejects it with a `PRCCNVRM`. Data with length greater than `MEDDTASZ` will be sent in Mode

X'03'. A value equal to SMLDTASZ means that no data is to be sent in Mode X'02', while a value equal to 2147483647 means that no data is to be sent in Mode X'03'. If both SMLDTASZ and MEDDTASZ have a value of zero, it means that no data is to be sent in Mode X'01' or Mode X'02'. If MEDDTASZ is not specified, the server assumes a default value of 1048576. MEDDTASZ is ignorable unless DYNDTAFMT is also set to TRUE.

**DF4** In Mode X'03' of Dynamic Data Format, the data is represented by an 8-byte reference called the *progressive reference*, whose life is tied to its originating cursor; i.e., if the cursor is closed implicitly or explicitly, the progressive reference will also be freed. The name "progressive" indicates that the data returned through such reference is always progressive or sequential, and it can only be used with the DDM command GETNXTCHK.

**DF5** This rule governs the content of the QRYDTA and EXTDTA representing an FD:OCA Generalized String when Dynamic Data Format is enabled. The FD:OCA Generalized String header, which is made up of a mode byte and a length of the data portion, is placed in the QRYDTA for each FD:OCA Generalized String column.

If the value of the column can be determined at the time the FD:OCA Generalized String header is generated in the QRYDTA, the following applies:

1. If the value is null for a nullable column, there is no FD:OCA Generalized String header nor data following the null indicator.
2. If the value has a length of zero, either Mode X'01' or X'02' can be used. An FD:OCA Generalized String header is generated with a mode byte value of X'01' or X'02' and a length of zero, and there is no data in QRYDTA nor EXTDTA for the column. The server determines whether X'01' or X'02' is more appropriate for the mode based on its implementation, and the application requester is required to handle both cases.
3. If the value has a length greater than zero and less than or equal to the SMLDTASZ specified on OPNQRY or EXCSQLSTT, Mode X'01' is used. An FD:OCA Generalized String header is generated with a mode byte value of X'01' and a length for the value bytes, and the value bytes are flown immediately following the FD:OCA Generalized String header in the QRYDTA.
4. If the value has a length greater than SMLDTASZ and less than or equal to the MEDDTASZ specified on OPNQRY or EXCSQLSTT, Mode X'02' is used. An FD:OCA Generalized String header is generated with a mode byte value of X'02' and a length for the value bytes, and the value bytes are flown in an associated EXTDTA.
5. If the value has a length greater than MEDDTASZ and the server determines that a progressive reference can be generated for the column data, Mode X'03' is used. An FD:OCA Generalized String header is generated with a mode byte value of X'03' and the length for the value bytes retrievable through the progressive reference, and the progressive reference is flown in the QRYDTA immediately following the FD:OCA Generalized String header. Using the DDM command GETNXTCHK with the progressive reference, the actual data is retrieved.
6. If the value has a length greater than MEDDTASZ but the server determines that a progressive reference cannot be generated for the column data, Mode X'02' is used. An FD:OCA Generalized String header is generated with a mode byte value of X'02' and a length for the value bytes, and the value bytes are flown in

an associated EXTDTA.

If the value of the column cannot be determined at the time the FD:OCA Generalized String header is generated in the QRYDTA, an FD:OCA Generalized String header containing a mode byte value of X'02' and a length having the unknown length indicator (the high-order bit) set to 1 is generated, the value bytes are flown in an associated EXTDTA.

**DF6** This rule governs the content of the QRYDTA and EXTDTA representing an FD:OCA Generalized String when Dynamic Data Format is not enabled. The FD:OCA Generalized String header, which is made up of a mode byte and a length of the data portion, is placed in the QRYDTA for each FD:OCA Generalized String column.

If the value of the column can be determined at the time the FD:OCA Generalized String header is generated in the QRYDTA, the following applies:

1. If the value is null for a nullable column, there is no FD:OCA Generalized String header nor data following the null indicator.
2. If the value has a length of zero, Mode X'02' is used. An FD:OCA Generalized String header is generated with a mode byte value of X'02' and a length of zero, and there is no data in QRYDTA nor EXTDTA for the column.
3. Otherwise, Mode X'02' is used. An FD:OCA Generalized String header is generated with a mode byte value of X'02' and a length for the value bytes, and the value bytes are flown in an associated EXTDTA.

If the value of the column cannot be determined at the time the FD:OCA Generalized String header is generated in the QRYDTA, an FD:OCA Generalized String header containing a mode byte value of X'02' and a length having the unknown length indicator (the high-order bit) set to 1 is generated, the value bytes are flown in an associated EXTDTA.

**DF7** This rule governs the content of the FDODTA and EXTDTA representing an FD:OCA Generalized String. The FD:OCA Generalized String header, which is made up of a mode byte and a length of the data portion, is placed in the FDODTA for each FD:OCA Generalized String column.

If the value of the column can be determined at the time the FD:OCA Generalized String header is generated in the FDODTA, the following applies:

1. If the value is null for a nullable column, there is no FD:OCA Generalized String header nor data following the null indicator.
2. If the value has a length of zero, Mode X'02' is used. An FD:OCA Generalized String header is generated with a mode byte value of X'02' and a length of zero, and there is no data in FDODTA nor EXTDTA for the column.
3. Otherwise, Mode X'02' is used. An FD:OCA Generalized String header is generated with a mode byte value of X'02' and a length for the value bytes, and the value bytes are flown in an associated EXTDTA.

If the value of the column cannot be determined at the time the FD:OCA Generalized String header is generated in the FDODTA, an FD:OCA Generalized String header containing a mode byte value of X'02' and a length having the unknown length indicator (the high-order bit) set to 1 is generated, the value bytes are flown in an associated EXTDTA.

- DF8** If the length in the FD:OCA Generalized String header does not have the unknown length indicator (the high-order bit) set to 1, it must match the actual length of the value bytes in the QRYDTA for Mode X'01', the value bytes in EXTDTA for Mode X'02', or the value bytes retrievable from the progressive reference for Mode X'03'; otherwise, an error will be returned to the application.

## 7.9 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 (TYPDEFNOVR) 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](#) 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.

**Note:** There are no restrictions on the data representation format chosen by the application server or application requester. In order to support a format different from the preferred format of the machine, conversion may be required at the sender. The receiver is still responsible for any data transformation needed to support the sender's format.

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 basic floating point number (BF16, BF8, BF4), 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 basic floating point number (BF16, BF8, BF4) — such as S/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 TYPDEFNAM, 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 CCSIDDBC, CCSIDMBC, or CCSIDXML 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 Generalized String header 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 $\leq 99999$ ).
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 scalar function on the column to extract the desired portion of the complete structure, in which case the DRDA type would be that of the function and not be the DATALINK type.

**DT21** A data representation transformation error (no representation of the character in the coded character set) may occur when an application requester or application server transforms string values to the sending data format representation as defined by the overrides (TYPDEFOVR) sent on ACCRDB or ACCRDBRM. The application program receives an SQLSTATE of 22021 for this error.

**DT22** The term “XML String” in DRDA refers to XML data in its serialized form. There is no restriction placed on the type of XML data that may be transported by the DRDA XML String types. For complete details on the format of serialized XML data, refer to World Wide Web Consortium ([www.w3.org/TR/xslt-xquery-serialization](http://www.w3.org/TR/xslt-xquery-serialization)).

Unlike traditional character data where the CCSID associated with the data is externally specified, XML by definition contains the encoding information within the data (see Section 4.3.3 of [www.w3.org/TR/1998/REC-xml-19980210](http://www.w3.org/TR/1998/REC-xml-19980210)). To support both the traditional SQL encoding mechanism and properly formed XML data, DRDA provides two XML types: XML String Internal Encoding to support XML consisting of its own encoding, and XML String External Encoding for XML requiring an external CCSID to be associated to provide the encoding information.

**DT23** The DRDA type XML String External Encoding provides a mechanism to override the default encoding as defined by XML (see Section 4.3.3 of [www.w3.org/TR/1998/REC-xml-19980210](http://www.w3.org/TR/1998/REC-xml-19980210)). This is accomplished by associating an external CCSID with the XML data. The external CCSID associated with this type is defined by the CCSIDXML specified on ACCRDB/ACCRDBRM, in a TYPDEFOVR or an explicit MDD SDA. It is important to note that any internal encoding declaration, if present, is ignored by DRDA processing. Therefore, it is possible that any internal encoding declaration present may become invalid if conversion occurs from one external CCSID to another.

**DT24** When communicating with systems at SQLAM Level 7 or lower, booleans are converted to 2-byte integers (I2 and NI2) in both directions.

## 7.10 RDB-Initiated Rollback (IR Rules)

**IR1** If the local environment at the application server attempts to initiate a global rollback when the server detects a relational database-initiated rollback, it must send an ABNUOWRM (RDB-initiated rollback) for an unprotected connection or a CMMRQSRM (type set to rollback) for a protected connection as part of the response to the request.

However, if the request is an OPNQRY, EXCSQLSTT, or CNTQRY using the Limited Block Protocol, the response may have to be deferred until the next CNTQRY, adhering to Rules QT2 through QT4 (see [Section 7.22.4](#) (on page 490)). If the application server defers the ABNUOWRM (RDB-initiated rollback) for an unprotected connection or the CMMRQSRM (type set to rollback) for a protected connection, it must return the message as the response to the next command, 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.

## 7.11 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 a reply message 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.

## 7.12 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,<sup>57</sup> CLOSE, COMMIT, CONNECT, ROLLBACK, RELEASE, SET CONNECTION, DISCONNECT, BEGIN DECLARE SECTION, END DECLARE SECTION, and local statements.<sup>58</sup>
- 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 variable appears in the SQL statement.

---

57. A connection using distributed unit of work protocols, the FETCH statement can flow to distributed application servers during the bind process. (See [Section 7.12](#) (on page 454)).

58. 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.

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 *:Hs* are inserted into the SQL statement for each element of the structure or array. Commas separate these *:Hs* (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 *:Hs* (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.
- PB29** BNOPT should not be used to flow bind options and values for which codepoints are explicitly defined in DRDA. For example, do not use BNOPT 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.

### 7.13 SQL Diagnostics (SD Rules)

- SD1** Statement-level SQL diagnostics is requested when accessing a remote RDB. Support for SQL diagnostics is optional.
- SD2** If a server does not support SQL diagnostics, the diagnostics group is returned as a null group. Any specific field not supported by the target server is set to null, an empty string, or a 0 value.
- SD3** The SQLCODE and SQLSTATE in the last condition must match the SQLCODE field and the SQLSTATE field defined in the SQLCAGRP.
- SD4** The diagnostics connection array is returned in the SQLCARD if and only if the target server is acting as an intermediate server or when the target server returns an optional SQLCARD as reply data to an ACCRDB command.
- SD5** Diagnostics returned in query blocks only contain conditions generated fetching the row. The query block contains a null statement and connection information. Statement and connection information is provided in a separate block SQLCARD. The block SQLCARD must follow the QRYDTA. If a row contains a query terminating SQLCARD, then the block SQLCARD is not required. Block SQLCARD rules:
- Block SQLCARD contains a null condition array and must be chained after the first QRYDTA.
  - Block SQLCARD precedes any EXTDTA externalized objects even when the query block contains a query terminating condition.
  - Block SQLCARD is returned if and only if diagnostics contains a statement group or a connection array.
  - Block SQLCARD is required to be returned once per cursor instance.
- SD6** Diagnostics are required to be returned when an SQL statement generates multiple error or warning conditions per command. The diagnostics group is required for cursors that support rowset processing or statements that contain array data. In these cases, multiple error and warning conditions are returned in the SQLCARD diagnostics group.

## 7.14 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](#)

**SE3** If user identification and authentication security is not provided using SECMGR Level 5 and above, 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](#)

**SE5** If SECMGR is at Level 5 and above, the application requester and application server must support at least one of the security mechanisms defined in [Chapter 10](#) (on page 515).

**SE6** Connections using the DCE security mechanism do not use GPSS channel bindings.

**SE7** If the application server does not support the SECMEC requested, then the application server must return the list of SECMEC values that it supports. The application requester must select one of the SECMEC combinations to use. If the application requester does not support any of the SECMEC values returned, then the application requester must terminate the network connection and return a security error to the user.

**SE8** If the application server supports the SECMEC but does not support the ENCALG or ENCKEYLEN requested, then the application server must return one or more values in ENCALG and ENCKEYLEN that it supports for the requested SECMEC. The application requester must select one of the ENCALG and ENCKEYLEN combinations to use. If the application requester does not support any of the ENCALG or ENCKEYLEN values returned, then the application requester must terminate the network connection and return a security error to the user.

**SE9** With the data encryption SECMECs, the sender must always encrypt the security-sensitive objects, SQLDTA, SQLDTARD, SQLSTT, SQLSTTVRB, SQLATTR, SQLDARD, SQLCINRD, SQLRSLRD, QRYDTA, EXTDTA, and SECTKNOVR. When encrypting the security-sensitive objects, the entire DSS containing the security-sensitive objects must be encrypted.

If the security-sensitive objects are not encrypted, then the PRCCNVCD value of '24'X must be returned.

**SE10** If the RDB is accessed as trusted, the following security mechanisms are restricted:

- An encrypted security mechanism is not supported unless the SECMEC instance variable on the initial ACCSEC command immediately following the EXCSAT command used was one of the encrypted security mechanisms. If this doesn't occur, the SECCHK command is rejected with a PRCCNVCD value of '101'X

indicating that the SECCHK is not supported.

Initial ACCSEC/SECCHK Security Mechanism	Subsequent SECCHK Security Mechanisms Allowed
USRENCPWD	USRIDPWD USRIDONL USRENCPWD
USRIDPWD USRIDONL USRIDNWPWD DCESEC KERSEC	USRIDPWD USRIDONL DCESEC KERSEC
EUSRIDONL EUSRIDPWD EUSRIDNWPWD	EUSRIDONL EUSRIDPWD EUSRENCPWD
EUSRIDDTA EUSRPWDDTA EUSRNPWDDTA	EUSRIDDTA EUSRPWDDTA
USRSBSPWD USRSSBPWD	USRIDONL USRSBSPWD USRSSBPWD
PLGIN	PLGIN

**Table 7-1** Compatible SECMECs

If the SECMECs are not compatible during a subsequent SECCHK command, the SECCHK command is rejected with a PRCCNVCD value of '101'X.

- A SECCHK command is allowed without a preceding ACCSEC command if the previous ACCRDBRM indicated a trusted connection. If a SECCHK command is received without the prior ACCRDBRM TRUST instance variable being set to commands are required prior to the SECCHK command if the ACCRDBRM did not indicate a trusted connection was established.

**SE11** Security Manager Level 8 allows the use of the shared private key for the duration of the network connection. The key is not regenerated on subsequent SECCHK commands on a trusted connection or on subsequent ACCSEC commands that do not contain the SECTKN parameter. The absence of a SECTKN parameter on the ACCSEC command for connections that have already established an encryption key from the previous use of the physical connection indicates to the server to reuse the shared private key to encrypt and decrypt objects on the connection.

## 7.15 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 [Section 7.12](#) (on page 454)). 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 [Section 7.12](#) (on page 454)), 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. Specifically, both the mandatory *pkgnamcsn* and the optional *cmdsrcid* parameters as specified on the OPNQRY command are used to uniquely identify a query. A successfully opened cursor constitutes a unique instance of the query as identified by a query instance identifier which is returned by the server in the OPNQRYRM reply. Subsequently, all query commands (CNTQRY or CLSQRY) issued against the server related to this cursor must also include the query instance identifier in order to uniquely identify the cursor. The optional *cmdsrcid* parameter, if specified for such operations, must be identical to what was specified previously on the EXCSQLSTT or OPNQRY command. Furthermore, the *cmdsrcid* parameter must also be identical to what was specified on the PRPSQLSTT command for a dynamically prepared query. See also Rules QI1 through QI5 in [Section 7.22.2](#) (on page 482).
- 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 [Section 7.12](#) (on page 454).) The application server can process or discard any statement with a duplicate section number that is subsequently received.

As stated in Rule SN3, once a cursor has been opened, for any subsequent query command (CNTQRY or CLSQRY) related to this cursor, the application requester must also send the query instance identifier to the server in order to uniquely identify the cursor. In addition, the optional *cmdsrcid* parameter, if specified for any such operation, must be identical to what was specified previously on the EXCSQLSTT or OPNQRY command. Furthermore, the *cmdsrcid* parameter must also be identical to what was specified on the PRPSQLSTT command for a dynamically prepared query. See also Rules QI1 through QI5 in [Section 7.22.2](#) (on page 482).

**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.

**SN10** Once assigned to a particular application source on the database connection by the application requester, the command source identifier remains in effect for the life of the application source and the application requester must specify this parameter either implicitly or explicitly on all the following commands: CLSQRY, CNTQRY, DSCSQLSTT, EXCSQLIMM, EXCSQLSTT, OPNQRY, and PRPSQLSTT. As such, the command source identifier and the *pkgnamcsn* parameters together uniquely identify a particular invocation of a command. No error is necessarily issued by the server if the application requester fails to specify the correct command source identifier on a command stemming from a particular application source because the server will be misled to handle the command as being from another application source altogether.

For additional information on how the command source identifier affects cursor operations, refer to Rules SN3, SN8, and SN11 in this section.

**SN11** If an intermediate server is connected to a downstream server that is operating at SQLAM Level 6 or below, the former is responsible for mapping requests from the upstream requester and replies from the downstream server so that both flows conform to the DRDA level at which the receiver is operating.

If explicitly specified, the *command source identifier* must be stripped off by the intermediate server before the command is forwarded on to the downstream server. Furthermore, the intermediate server must allow no more than one *command source identifier* value to be specified either explicitly or implicitly by the upstream requester for a database connection. In essence the first *command source identifier* specified explicitly or implicitly on a command on the database connection is also the only one that is supported. The intermediate server must therefore reject any command on the existing database connection bearing another *command source identifier* value either explicitly or implicitly. The method used to reject the command depends on the command:

- If the command is OPNQRY, then the OPNQFLRM reply message is returned with an SQLCARD specifying 56072.

- For any other command, then an SQLCARD is returned specifying 56072.

Also see Rule QI5 in [Section 7.22.2](#) (on page 482).

## 7.16 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 than 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.
- SP5** An EXCSQLSTT command in an atomic chain enclosed within a BGNATMCHN-ENDATMCHN command pair must not be for a CALL statement for a stored procedure. If the application server receives such a command, it should terminate processing of the atomic chain with SQLSTATE '560B6'.
- SP6** If the value of the *rtnsqlda* and *typsqlda* parameters on the execute SQL Call statement command request descriptive information for stored procedure parameters, the SQL descriptor area reply data is returned before the SQL data reply data. The *rtnsqlda* parameter is ignored if no SQLDTARD is returned describing the returned parameters.

## 7.17 SET Statement (ST Rules)

- ST1** Local SET statements are SET statements that do not flow to the server. The following are local SET statements:
- SET CONNECTION
  - SET CURRENT PACKAGESET
- All other or unrecognized SET statements are considered non-local. Non-local SET statements may flow with the BNDSQLSTT, EXCSQLSET, EXCSQLIMM, and EXCSQLSTT commands.
- 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.
- ST5** The SET CURRENT PACKAGE PATH statement is a special type of non-local SET statement that can only flow with the EXCSQLSET command. The requester manages the CURRENT PACKAGE PATH value, and propagates the value to the server in the event that the value has been changed by the application since the last request was sent. In this case, the value will flow to the server prior to sending the next remote SQL request. The EXCSQLSET command is used to propagate the setting of the CURRENT PACKAGE PATH value. The value cannot be sent as part of the PREPARE or EXCSQLIMM commands. Any attempt to send this value as part of these commands will result in the SQLSTATE 42612.
- ST6** When a non-local SET statement is executed, only the execution environment identified by the value of the *cmdsrid* parameter as specified on the command is affected.
- ST7** When a server returns a group of SET statements in order to allow the connection to be reused for another application on a transaction boundary as per Rule CU17 in [Section 7.6](#) (on page 436), the special register settings apply solely to the execution environment identified by the default *cmdsrid* parameter value of 0.
- ST8** When a server returns a group of SET statements as per the setting of the *rtinsetstt* parameter on the command, the special register settings apply solely to the execution environment identified by the *cmdsrid* parameter as specified on the command.

## 7.18 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](#)
- [Section 13.6.2.4](#)

**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](#)
- [Section 13.6.2.4](#)

**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](#)

**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.

## 7.19 Update Control (UP Rules)

- UP1** If the application is not using the services of a sync point manager or XA 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. If the RDBUPDRM reply message reports an unprotected update has been performed downstream via the *unpupd* parameter, in violation of the *unpupdaltw* parameter as specified on the previous ACCRDB command, the application requester must roll back the unit of work.
- In particular, no more than one RDBUPDRM reply message can be sent back by the server in the reply chain in response to a DDM command. The application requester must report SQLSTATE 58009 to the application if this is not the case.
- The sending and receipt of RDBUPDRM is not supported when using SQLAM Level 3.
- UP4** If there are multiple DDM reply messages and/or data objects in response to a DDM command of which one is an RDBUPDRM, the RDBUPDRM must be at the beginning of the reply chain if the reply does not contain an OPNQRYRM reply message or a QRYDTA reply data object.
- UP5** If there are multiple DDM reply messages and/or data objects in response to a DDM command of which one is an RDBUPDRM, the RDBUPDRM must be at the beginning or the end of the reply chain if the reply contains an OPNQRYRM reply message and/or a QRYDTA reply data object.
- If located at the end of the reply chain, the RDBUPDRM reply message must still come before the MONITORRD reply data object if it is present.

## 7.20 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.

## 7.21 Names

The following sections define the rules for end-user names, SQL object names, relational database names, and target program names.

### 7.21.1 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 printable characters. Some security managers require end-user names to be in uppercase or in lowercase, and some allow mixed-case end-user names. The receiver of an end-user name is required to fold the end-user name to the appropriate case prior to authenticating the end-user name by the local security manager. End-user names must be in the character set identified by the CCSIDMGR used for the connection.

### 7.21.2 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](#) 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 255 bytes. The maximum length of an OBJECTID is 255 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 255 bytes. The maximum length of a PACKAGEID is 255 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 and cannot be zero.

**ON1D** The fully qualified name for a stored procedure is RDB\_NAME.COLLECTION.PROCEDURE. The maximum length of COLLECTION is 255 bytes. The maximum length of a PROCEDURE is 255 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.

### 7.21.3 Relational Database Names (RN Rules)

**RN1** The first six bytes of an RDB\_NAME must be registered with The Open Group. See [Section 6.2](#) for an explanation of how to register RDB\_NAMES. The first two bytes are a country code defined in *ISO 3166*. The characters of the country code are chosen from the uppercase letters (A through Z). The next four bytes are an owning enterprise code of the enterprise registering the first six bytes of the RDB\_NAME. The owning enterprise code must be chosen from the uppercase letters (A through Z) and the numerics (0 through 9).

The remaining bytes of an RDB\_NAME have 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 255 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](#)
- [Section 13.6.2.5](#)

**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](#)
- [Section 13.6.2.5](#)

**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](#)
- [Section 13.6.2.5](#)

#### 7.21.4 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](#)
- [Section 13.6.2.6](#)

**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](#)
- [Section 13.6.2.6](#)

**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](#)
- [Section 13.6.2.6](#)

**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](#)
- [Section 13.6.2.6](#)

**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](#)
- [Section 13.6.2.6](#)

**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](#)
- [Section 13.6.2.6](#)

**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](#)
- [Section 13.6.2.6](#)

**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](#)
- [Section 13.6.2.6](#)

## 7.22 Query Processing

For query processing, there are rules for blocking, query data transfer protocols, and terminating, interrupting, continuing query data or result set transfer, and query instances.

### 7.22.1 Blocking

*Blocking* refers to the process of sending query reply data in units known as *query blocks*. Each query block is a QRYDTA OBJDSS.<sup>59</sup> The purpose of blocking is to allow the application requester to pace the amount of query data it receives for query replies that contain more than one row. Other methods in DRDA are used in conjunction with blocking to control how many rows are returned with a reply.

Each query block is a QRYDTA object containing query data returned in reply to a command that requests such data. On the command, the application requester specifies the *qryblksz* parameter that is to apply to each query block returned for the command. The actual size of each query block returned depends both on the query block size specified by the application requester, the type of query blocks the server chooses to return, the size of the rows returned, and the number of rows returned. The server indicates the type of query blocks it will return in the OPNQRYRM. If the server chooses to return exact query blocks, then every query block is exactly the specified query block size, except for possibly the last query block which may be shorter. If the server chooses to return flexible query blocks, then every query block is at least as large as the specified query block size, except for possibly the last query block which may be shorter. In the case of flexible query blocks, the size of the QRYDTA may be greater than the query block size limit if the query block must be expanded beyond its initial size to contain a complete row.<sup>60</sup> A flexible query block can only be expanded beyond its initial size once. The size of the last (or only) QRYDTA may be less than the specified query block size if the number of rows returned is less than the maximum number of rows that can fit into the query block.

The blocking rules are based on a minimum block size of 512 bytes (see Rule BS2 in [Section 7.22.1.2](#) (on page 477)). 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.

---

59. In SQLAM Level 6 and below, blocking refers to the blocking of all query reply objects, including OPNQRYRM, SQLCINRD, QRYDSC, QRYDTA, ENDQRYRM, and SQLCARD.

60. In SQLAM Level 6 and below, the query block size limit is exact and every query block, except the last one, must be filled completely and must have exactly the size specified by the query block size parameter specified by the application requesters.

## 7.22.1.1 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:

**BF21** Blocking refers to the construction of reply objects containing data from one or more rows of a query answer set where the size of the generated object is governed by a size parameter.

Only QRYDTA objects are governed by this size parameter and are said to be blocked when they adhere to the rules governing the size of the object. When discussing a QRYDTA object as a blocked object, it is called a *query block*.

A query block can be one of two types: flexible or exact. Additional rules governing exact query blocks and flexible query blocks are given in the remaining BF Rules.

The application server chooses the type of query blocks to be returned for a query and indicates its choice on the OPNQRYRM reply message, with the following restriction:

The application server may only return flexible blocks for rowset cursors (that is, if QRYATTSET=TRUE).

**BF22** Blocking applies to query answer set data as follows:

Query answer set data consists of base row data and optional externalized row data.

All base row data flows in QRYDTA objects. Base row data for an answer set row consists of two nullable groups, the first containing the SQLCARD and the second containing the base row data values. Either or both of these groups may be null.

Externalized row data flows in EXTDTA objects according to Rules CH3 through CH5 (see [Section 7.22.1.3](#) (on page 478)). EXTDTA data objects are not governed by the size parameter and thus are not blocked.

Base row data for an answer set row includes each column in the answer set row. If the answer set row contains a non-FD:OCA Generalized String column, the column data itself is included in the base row data. If the answer set row contains an FD:OCA Generalized String column, the FD:OCA Generalized String header is included in the base row, while the data portion may flow immediately following the header in the QRYDTA or in an associated EXTDTA object as externalized row data. See Data Format (DF Rules) in [Section 7.8](#) for more details.

An answer set row may consist only of base row data or may consist of base row data and externalized data. A base row is complete when the base row data for an answer set row has been sent. An answer set row consisting only of base row data is complete when the base row is complete; that is, when all the 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 the associated EXTDTAs for the row have been sent.

Only columns that are FD:OCA Generalized Strings may be externalized. All other data types must flow as base row data.

**BF23** The base row data for an answer set row or an SQL rowset must be completely contained in a flexible query block when flexible blocking is in effect for the query.

**BF24** The base row data for an answer set row may span exact query blocks when exact blocking is in effect for the query.

**BF25** If an exact query block ends with a partial row of data that does not contain the end of that row, the partial row must fill the remaining space in the query block.

**BF26** In the case of single row fetch, if one or more query blocks are returned as a reply to a command, then the query block(s) must contain either the completion of a partial row sent in exact query blocks for a previous command or they must contain the complete base row data for at least one answer set row.

If the last query block returned has space for additional row data, then additional row data can be added to the query block according to Rules BF27 and BF28.

**BF27** The initial size of a flexible query block is given by Rule BS1 in [Section 7.22.1.2](#) (on page 477). Aside from the DSS header and the length and codepoint fields for the QRYDTA object, all other space in the query block is available to contain base row data.

In the case of single-row fetch, one or more rows are retrieved from the relational database and the base row data for the retrieved rows are added to the space remaining in the flexible query block as follows:

A base row can be retrieved and added to the flexible query block as long as the flexible query block can contain any part of a base row. If the complete base row can be added to the flexible query block, then the row is added to the flexible query block and the space remaining can be used to contain one or more additional base rows. If the space remaining in the flexible query block cannot contain the complete base row, the flexible query block is expanded beyond its initial size to contain the complete base row and the row thus added is the last row that can be added to the flexible query block. Additional rows may be retrieved and added to extra query blocks as allowed by the *maxblkext* parameter. If the flexible query block does not have room for any part of an additional base row, then no additional row should be retrieved from the relational database for inclusion in this flexible query block.

In the case of multi-row fetch against a rowset cursor, a single SQL rowset is retrieved from the relational database and the base row data for the retrieved rows in the SQL rowset is added to the flexible query block. If the initial query block size for the flexible query block is too small to contain the entire SQL rowset, the flexible query block is enlarged to contain the entire SQL rowset.

**BF28** The size of an exact query block is given by Rule BS1 in [Section 7.22.1.2](#) (on page 477). Aside from the DSS header and the length and codepoint fields for the QRYDTA object, all other space in the query block is available to contain base row data.

In the case of single row fetch, one or more rows are retrieved from the relational database and the base row data for the retrieved rows are added to the space remaining in the exact query block as follows:

If a partial row was not returned by a previous command, a base row can be retrieved and added to the exact query block as long as space is available in the remainder of the exact query block to contain any part of a base row. If the complete base row can be added to the exact query block, then the row is added to the exact query block and the space remaining can be used to contain one or more additional base rows. If the space remaining in the query block cannot contain the complete base row, only that part of the row data that can fit in the remainder of the exact query block is added to the query block. If this is the first row added to the exact query block, then additional exact query blocks are generated to contain the remainder of the base row data. If this is not the first row added to the exact query block, then either a partial row is returned in the last exact query block or extra query blocks can be generated to contain the remainder of the row (and possibly, additional rows) as allowed by the *maxblkext* parameter. If the

exact query block does not have room for any part of an additional base row, then no additional row should be retrieved from the relational database inclusion in this flexible query block.

If a partial row was returned by a previous command, then the remainder of the row is added to the exact query block. If the remainder of the row is not completely contained in the query block, then additional exact query blocks are generated to contain the remainder of the base row data. Additional row data can be added to the space remaining in the exact query block that contains the end of the row as described above.

**BF29** If more than one flexible query block is returned for a query or result set in response to a command, then the size of each flexible query block, except possibly the last one, must be at least as large as the initial query block size specified on the command.

If the command is EXCSQLSTT, then this rule applies separately for each result set component returned.

**BF30** If more than one exact query block is returned for a query or result set in response to a command, then the size of each exact query block, except possibly the last one, must be exactly equal to the query block size specified on the command.

If the command is EXCSQLSTT, then this rule applies separately for each result set component returned.

For additional description of possible block formats, see the DDM terms OPNQRY, CNTQRY, EXCSQLSTT, LMTBLKPRC (Limited Block Protocol), and FIXROWPRC (Fixed Row Protocol).

## 7.22.1.2 Block Size (BS Rules)

**BS1** The application requester specifies the size value that governs the generation of a query block by means of a parameter on the OPNQRY, CNTQRY, and EXCSQLSTT commands. Query block size may change on any or each CNTQRY request.

For flexible query blocks, the size value specifies an initial size for the query block. The size of a flexible query block can expand beyond its initial size.

For exact query blocks, the size value specifies the exact size for the query block.

If a MGRLVLOVR object precedes the QRYDSC object, then Rule CU28 applies.

**BS2** The minimum block size parameter value is 512 bytes.

**BS3** The maximum block size parameter value is 10,485,760 (10M).

## 7.22.1.3 Chaining (CH Rules)

**CH1** The DDM RPYDSS or OBJDSS objects returned for an open query or result set as the replies to an OPNQRY, CNTQRY, or EXCSQLSTT command are chained together in a fixed order.

In the non-error case, the replies returned in response to an OPNQRY or EXCSQLSTT command for an open cursor or stored procedure result set are the OPNQRYRM RPYDSS and an optional SQLCARD OBJDSS, followed by the QRYDSC OBJDSS, followed by zero, one, or more objects containing answer set data, followed by zero, one, or more EXTDTAs. In addition, in response to the EXCSQLSTT command, the optional SQLCINRD OBJDSS may be returned between the OPNQRYRM and QRYDSC objects or between the SQLCARD OBJDSS and the QRYDSC if an optional SQLCARD is returned.

Table 7-2 gives the ordering of the maximal set of reply objects that can be returned in response to an EXCSQLSTT command that returns one result set with only one query block of data (the query-related reply objects, starting with OPNQRYRM through ENDQRYRM and SQLCARD repeat for additional result sets).

**Table 7-2** Maximal Example for EXCSQLSTT

DDM Object	DDM Carrier
RDBUPDRM	RPYDSS
TYPDEFNAM	OBJDSS
TYPDEFOVR	OBJDSS
SQLDTARD	OBJDSS
RSLSETRM	RPYDSS
TYPDEFNAM	OBJDSS
TYPDEFOVR	OBJDSS
SQLSTT	OBJDSS
OPNQRYRM	RPYDSS
TYPDEFNAM	OBJDSS
TYPDEFOVR	OBJDSS
SQLCARD	OBJDSS
TYPDEFNAM	OBJDSS
TYPDEFOVR	OBJDSS
SQLCINRD	OBJDSS
TYPDEFNAM	OBJDSS
TYPDEFOVR	OBJDSS
QRYDSC	OBJDSS
QRYDTA	OBJDSS
EXTDTA	OBJDSS
ENDQRYRM	RPYDSS
TYPDEFNAM	OBJDSS
TYPDEFOVR	OBJDSS
SQLCARD	OBJDSS
EXTDTA	OBJDSS
RDBUPDRM	RPYDSS

Note that only one of the two RDBUPDRM reply messages in Table 7-2 can exist in the reply chain as per the Update Control (UP Rules) in Section 7.19 (on page 467).

If the stored procedure has no parameters, then an SQLCARD is returned instead of the SQLDTARD and associated EXTDTA objects. Further, where the RDBUPDRM is shown, any one of the reply messages ENDUOWRM, RDBUPDRM, or CMMRQSRM may be returned. If RDBUPDRM is returned, either ENDUOWRM or CMMRQSRM may also be returned after the RDBUPDRM. Also note that the SQLSTT OBJDSS is repeated for each SQL SET statement that is returned, as per the setting of the *rinsetstt* parameter as specified on the EXCSQLSTT command.

Table 7-3 gives the ordering of the maximal set of reply objects that can be returned in response to an EXCSQLSTT command that returns one result set with only one query block of data (the query-related reply objects, starting with OPNQRYRM through ENDQRYRM and SQLCARD repeat for additional result sets).

**Table 7-3** Maximal Example for EXCSQLSTT

DDM Object	DDM Carrier
RDBUPDRM	RPYDSS
TYPDEFNAM	OBJDSS
TYPDEFOVR	OBJDSS
SQLDTARD	OBJDSS
RSLSETRM	RPYDSS
TYPDEFNAM	OBJDSS
TYPDEFOVR	OBJDSS
SQLRSLRD	OBJDSS
OPNQRYRM	RPYDSS
TYPDEFNAM	OBJDSS
TYPDEFOVR	OBJDSS
SQLCARD	OBJDSS
TYPDEFNAM	OBJDSS
TYPDEFOVR	OBJDSS
SQLCINRD	OBJDSS
TYPDEFNAM	OBJDSS
TYPDEFOVR	OBJDSS
QRYDSC	OBJDSS
QRYDTA	OBJDSS
ENDQRYRM	RPYDSS
TYPDEFNAM	OBJDSS
TYPDEFOVR	OBJDSS
SQLCARD	OBJDSS
EXTDTA	OBJDSS
RDBUPDRM	RPYDSS

Note that only one of the two RDBUPDRM reply messages in Table 7-3 can exist in the reply chain as per the Update Control (UP Rules) in Section 7.19 (on page 467).

If the stored procedure has no parameters, then an SQLCARD is returned instead of the SQLDTARD and associated EXTDTA objects. Further, where the RDBUPDRM is shown, any one of the reply messages ENDUOWRM, RDBUPDRM, or CMMRQSRM may be returned. If RDBUPDRM is returned, either of ENDUOWRM or CMMRQSRM may also be returned after the RDBUPDRM.

Each DDM object carried in an OBJDSS, except for QRYDTA and EXTDTA objects, may be carried in its own OBJDSS or combined with other objects to reduce the number of OBJDSSs sent.

Each QRYDTA object always flows in its own OBJDSS.

Each EXTDTA object always flows in its own OBJDSS. 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. The EXTDTA objects associated with a row in a QRYDTA may be chained to the query block(s) containing the row and flow in the same reply chain (for example, when *rtnextdta* is RTNEXTALL), or they may be returned as reply objects to a subsequent CNTQRY that returns only the EXTDTA (for example, when *rtnextdta* is RTNEXTROW).

**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 Generalized String header appear in the base row.

All EXTDTA objects associated with a row must flow before other EXTDTAs from subsequent rows can flow.

If the application requester 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. If the EXTDTA objects do not flow in the reply chain with the query block (for example, when *rtnextdta* is RTNEXTROW), then no subsequent extra query block can flow.

**CH5** If a base data object contains FD:OCA Generalized Strings, then any associated 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 objects that can be chained after an EXTDTA are another EXTDTA object or a QRYDTA object for the same query or a NULL SQLCARD if it is the last EXTDTA for the query and there are no more objects chained to this last EXTDTA. A query-terminating reply message may also be chained after an EXTDTA according to Rule QT5 (see [Section 7.22.4](#) (on page 490)).

If the command is a CNTQRY, and the query is completed by the command, and Rule QT5 allows an ENDQRYRM and chained SQLCARD to be added to the reply chain, then the ENDQRYRM is chained after the last EXTDTA for the last row that has EXTDTAs associated with it. If the EXTDTAs are chained to the associated QRYDTA object, then the ENDQRYRM and SQLCARD are chained to the last EXTDTA in the chain. If the EXTDTAs are returned in response to a CNTQRY according to the *rtnextdta* option of RTNEXTROW, then the ENDQRYRM reply message and its associated SQLCARD reply data object are chained to the last EXTDTA associated with the previously sent QRYDTA containing the associated base data.

The EXTDTA may not be associated with the last row returned in the QRYDTA. If an ENDQRYRM is sent after an EXTDTA that is not associated with the last base row in the associated QRYDTA, then the ENDQRYRM should be stacked by the application requester until the application has fetched the last row in the QRYDTA or terminates the query.<sup>61</sup>

If the command is an EXCSQLSTT for a stored procedure with query result sets, the EXTDTAs returned associated with the SQLDTARD containing the parameters must be the last objects in the reply chain, with the possible exception of an SQLCARD. They

follow the results set information objects and the query reply objects for the result set. No extra query blocks may be returned with any query result sets in this case.

- 
61. For example, the requester would need to stack the ENDQRYRM in the following cases.
1. Suppose the requester specified RTNEXTALL for the RTNEXTDTA option of CNTQRY. Suppose a QRYDTA contains the last five rows in a query where each row has one LOB column defined. Suppose only rows 1 and 2 have non-null, non-zero-length LOBs. Suppose that Dynamic Data Format is not enabled. Then the objects returned for the CNTQRY would be QRYDTA (containing five rows), followed by the EXTDTA for row 1, followed by the EXTDTA for row 2, followed by the ENDQRYRM and SQLCARD. The requester will get the base data for row 1 and the EXTDTA for row 1. The requester next gets the base data for row 2 and EXTDTA for row 2. In this case, the requester must stack the ENDQRYRM until all the base data in the QRYDTA (rows 3, 4, and 5) have been processed.
  2. Suppose the requester specified RTNEXTROW for the RTNEXTDTA option of CNTQRY. Suppose a QRYDTA contains the last five rows in a query where each row has one LOB column defined. Suppose only rows 1 and 2 have non-null, non-zero-length LOBs. Suppose that Dynamic Data Format is not enabled. Then the objects returned for the CNTQRY would be QRYDTA (containing five rows). The requester will get the base data for row 1 and send a CNTQRY to get the EXTDTA for row 1. The requester next gets the base data for row 2 and sends a CNTQRY to get the EXTDTA for row 2. In this case, the server may optionally return the ENDQRYRM and SQLCARD for the query since the last EXTDTA for the query has been returned. The requester must stack the ENDQRYRM until all the base data in the QRYDTA (rows 3, 4, and 5) have been processed.

### 7.22.2 Query Instances (QI Rules)

Each time a cursor is opened, a query instance is created. DRDA allows a cursor to be open concurrently from different invocations because each invocation results in a unique query instance of the cursor. This concept is explained in detail in [Section 4.4.6](#) (on page 145).

**QI1** If a query is opened successfully, the server must uniquely identify its instance with the *qryinsid* parameter on the OPNQRYRM reply message. If the server fails to do so, the application requester must report SQLSTATE '58009'.

**QI2** If the server is unable to generate a unique value for the *qryinsid* parameter when opening a query, the server must return the QRYPOPRM reply message in response to an OPNQRY command.

If instead the failure occurs when opening a query in response to an EXCSQLSTT command for a stored procedure call, the server must not return this query result set to the requester.

**QI3** No two query instances can have an identical *qryinsid* value if they also have identical *pkgnamcsn* values and identical *cmdsrcid* values. However, two queries on a single database connection (that is, their *pkgnamcsn* values and/or *cmdsrcid* values must not be identical) can possibly have query instances with an identical *qryinsid* value. For further information on the *cmdsrcid* parameter, see Rules SN3, SN8, SN10, and SN11 in [Section 7.15](#) (on page 460).

The server may reuse a *qryinsid* value for a new query instance once the current query instance which is associated with the *qryinsid* is closed for that application source.

**QI4** The application requester must identify the query instance on CNTQRY and CLSQRY commands with the *qryinsid* parameter. Failure to do so must result in the server returning SYNTAXRM with *synerrcd* set to X'0E'.

A DSCSQLSTT command operating on an SQL statement that has an open cursor associated with it must include the *qryinsid* parameter to indicate the instance of the query in use. Failure to do so may result in the server attempting to perform the describe operation on another SQL statement in some other context, which may or may not result in an error.

An EXCSQLSTT or EXCSQLIMM command for a positioned DELETE/UPDATE SQL statement must include the *qryinsid* parameter to indicate the instance of the query in use, unless only a single query instance exists for the section, in which case the *qryinsid* parameter is optional. Failure to do so may result in the server attempting to perform the execute operation on another SQL statement in some other context, which may or may not result in an error.

**QI5** If an intermediate server is connected to a target server that is operating at SQLAM Level 6 or below, the former is responsible for mapping requests from the application requester and replies from the target server so that both flows conform to the DRDA level at which the receiver is operating.

There can only be at most one open cursor per *pkgnamcsn* at the target server in this case. Therefore, once such a cursor has been opened, the intermediate server must generate a dummy *qryinsid* to return to the application requester for the only allowable query instance for this *pkgnamcsn*.

The intermediate server must subsequently ensure for such an open cursor, that the *qryinsid* specified on a CNTQRY or CLSQRY by the application requester matches the one that is generated at open time without exposing it to the target server. And if the application requester attempts to open another cursor with a *pkgnamcsn* for which there

is already an open cursor, the intermediate server must return a QRYPOPRM reply message to indicate that it is not possible to have another instance of the query.

Also see Rule SN11 in [Section 7.15](#) (on page 460).

### 7.22.3 Query Data Transfer Protocols (QP Rules)

#### QP1 Fixed Row Protocol

In the non-error case, the response to OPNQRY consists of the OPNQRY reply message (OPNQRYRM) and the FD:OCA description of the data (QRYDSC). If the cursor is scrollable and the cursor is not defined for rowset processing, the *qryrowset* parameter on the OPNQRY command controls whether data rows are to be returned after the QRYDSC. If the cursor is using rowset positioning, the *qryrowset* parameter is ignored on the OPNQRY command. If a non-zero *qryrowset* value is specified, then QRYDTA objects are returned according to Rule QP4. If no *qryrowset* value is specified, no data rows are returned. If a zero *qryrowset* value is specified, no data rows are returned. If the cursor is a rowset cursor supporting multi-row fetch, no data rows are returned in response to the OPNQRY.

Answer set data may not be returned with the OPNQRY response if the answer set contains any LOB columns, unless the OUTOVR OPT is set to OUTOVRNON and Dynamic Data Format is enabled, since the output format of those columns as either LOB data values or LOB locator values is not known until the first application FETCH request. If OUTOVR OPT is set to OUTOVRNON and Dynamic Data Format is enabled, since no OUTOVR can be specified on the CNTQRY command, the LOB columns are returned according to the Data Format (DF Rules); see [Section 7.8](#) for more details.

The application requester must use CNTQRY to retrieve (more) answer set data. In the case of non-scrollable cursors, the first CNTQRY command constitutes the initial retrieval of data from the answer set. In the case of rowset cursors, each CNTQRY command retrieves an SQL rowset and all the requested rows in the SQL rowset are returned with the command. For scrollable cursors, answer set data may have been returned with the OPNQRY command. If a CNTQRY command does not specify a *qryrowset* parameter, then the CNTQRY command causes at most one FETCH request to be performed at the application server and a successful FETCH retrieves exactly the number of rows of answer set data requested by the application. For rowset cursors, a CNTQRY command must specify a *qryrowset* parameter with a non-zero value.<sup>62</sup> If a CNTQRY command does specify a non-zero *qryrowset* parameter, then QRYDTA objects are returned according to Rule QP4. The answer set is transmitted in one or more OBJDSSs, each of which is a QRYDTA or an EXTDTA, that are chained together; the number of OBJDSSs depends on the number of rows returned, the size of the rows, and the number of EXTDTA objects to be sent.

EXTDTA objects associated with retrieved rows are sent in accordance with Rules CH3 and CH4 in [Section 7.22.1.3](#) (on page 478).

For non-scrolling cursors, the query is complete when a CNTQRY results in RPYDSS indicating end of query (ENDQRYRM) chained to an OBJDSS containing an SQLCARD data object. This can be the result of the server's decision to close a query implicitly when it has run out of rows (SQLSTATE 02000) based on some other cursor properties, and also on the value of the *qryclsimp* parameter that has previously been sent on the OPNQRY command.

Otherwise, if the server chooses not to close the query implicitly, the query is complete when the application requester closes the query explicitly by sending a CLSQRY command to the server, or when the transaction is rolled back.

---

62. Multi-row fetch is not supported in DRDA Level 1.

For cursors that scroll, the query is complete when the application closes the cursor. This results in the application requester flowing a CLSQRY command to the application server.

The DDM term FIXROWPRC more completely defines this protocol. See also Rules QT1, QT2, QT3, and QT4 in [Section 7.22.4](#) (on page 490).

For rowset cursors, a statement-level SQLCARD must be sent after each QRYDTA or after the ENDQRYRM if chained to the QRYDTA indicating the end of query. Row-level SQLCARDS in the QRYDTA must be null for rowset cursors.

## QP2 Limited Block Protocol

In the non-error case, the response to OPNQRY consists of the OPNQRY reply message (OPNQRYRM) and the FD:OCA description of the data (QRYDSC). Answer set data may also be returned by the application server if it is not explicitly prohibited from doing so by this or another rule.

Answer set data cannot be returned with the OPNQRY response if the answer set contains any LOB columns, unless the OUTOVROPT is set to OUTOVRNON and Dynamic Data Format is enabled, since the output format of those columns as either LOB data values or LOB locator values is not known until the first application FETCH request. If OUTOVROPT is set to OUTOVRNON and Dynamic Data Format is enabled, since no OUTOVR can be specified on the CNTQRY command, the LOB columns are returned according to the Data Format (DF Rules); see [Section 7.8](#) for more details.

If the application server exercises the option to return answer set data with the OPNQRY response, one or more query blocks contain the data, each query block being a QRYDTA OBJDSS that is chained to the previous reply object. The number of query blocks returned depends on the number of rows returned, the size of the rows, and the extra query block limits negotiated between the application requester and application server.

- For scrollable cursors, whether rows are returned and how many rows are returned depend on the *qryrowset* parameter specified on the OPNQRY command. If a non-zero *qryrowset* value is specified, then QRYDTA objects are returned according to Rule QP4. If no *qryrowset* value is specified, then an implicit *qryrowset* value is set according to Rule QP4 and QRYDTA objects are returned according to Rule QP4.
- For non-scrollable cursors, the *qryrowset* parameter can also control whether rows are returned and how many rows are returned with the OPNQRY command, but if it is not specified, there is no implicit rowset for the cursor. If no *qryrowset* value is specified, then one or more QRYDTA objects is returned according to the limits specified by the *maxblkext* parameter.
- If Dynamic Data Format is enabled, only RTNEXTDTA=RTNEXTALL is supported on the CNTQRY of the query; this setting is applied with OPNQRYRM allowing one or more QRYDTAs and all the EXTDTAs associated with the complete base rows to be flown following the QRYDSC.

The application requester must use CNTQRY to retrieve more answer set data. If answer set data is returned, then the following applies:

- If none of the answer set columns will flow as externalized FD:OCA data in EXTDTAs, the CNTQRY response consists of at least one QRYDTA containing row data for one or more rows according to the Block Formats (BF Rules) in [Section 7.22.1.1](#) (on page 474). If the application requester is capable of accepting

extra query blocks, then the application server may chain additional query blocks.

- If any of the answer set columns will 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. The CNTQRY command has one of the following responses:

- For the first CNTQRY, or a subsequent CNTQRY retrieving additional base row data, the application server returns at least one query block and the base row data that completes at least one row. The QRYDTA may contain additional rows according to the Block Formats (BF Rules) in [Section 7.22.1.1](#) (on page 474).

If the application requester specified that all EXTDTAs are to be returned with the base data, then the EXTDTA objects for all complete rows in the query block are returned following the query block. The next CNTQRY command retrieves additional base row data along with any associated EXTDTAs.

If the application requester specified that EXTDTA objects are to be returned a row at a time, no EXTDTAs are returned with the base data in the QRYDTA. 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 FD:OCA Generalized Strings or FD:OCA Generalized Strings with zero lengths or FD:OCA Generalized Strings that have no associated EXTDTA objects. 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 EXTDTA objects corresponding to the FD:OCA Generalized String headers in the base row. This rule only applies if EXTDTA objects are to be returned a row at a time.

For non-scrollable cursors or non-scrollable result sets, 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 Rule CH5 (see [Section 7.22.1.3](#) (on page 478)) applies to the RPYDSS. For cursors or result sets 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 LMTBLKPRC more completely defines this protocol. See Rules QT1, QT2, QT3, and QT4 in [Section 7.22.4](#) (on page 490).

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.

**QP4** This rule applies to scrollable or non-scrollable non-rowset cursors when an explicit positive *qryrowset* parameter is specified on an OPNQRY, CNTQRY, or EXCSQLSTT command that returns stored procedure result sets. It also applies to scrollable non-rowset cursors when an implicit positive *qryrowset* value is required. If an implicit *qryrowset* value is required, the value used is 64. This is an arbitrarily-chosen architectural constant that allows both the application requester and application server to know the size of an implicit rowset when no explicit *qryrowset* value is specified.<sup>63</sup> See Rule QP2 for the case when this is needed.

In both cases, the application server is required to send a DRDA rowset to the application requester. The DRDA rowset has a size, defined to be either the explicit value specified on the command or the implicit value determined above. When an implicit value is used, the DRDA rowset is said to be an implicit DRDA rowset.

When a DRDA rowset of size *S* is to be returned in response to an OPNQRY command, the application server performs at most *S* single-row fetches to populate the rowset. The first row in the DRDA rowset returned by the application server consists of the first row in the result table, followed by the next *S*–1 rows in sequence (FETCH NEXT) in the result table. The rows in the DRDA rowset are returned as indicated in Rule QP2, specifying how answer set data is to be returned in response to an OPNQRY. If the DRDA rowset is implicit, then any *maxblkext* value on the OPNQRY command is ignored and a *maxblkext* value of zero is used, indicating that no extra query blocks are to be returned.

When a DRDA rowset of size *S* is to be returned in response to a CNTQRY command, the application server performs at most *S* single-row fetches to populate the DRDA rowset. The first row in the DRDA rowset consists of the row identified by the navigational parameters on the CNTQRY command, followed by the next *S*–1 rows in sequence (FETCH NEXT) in the result table. The rows in the DRDA rowset are returned as indicated in Rule QP2, specifying how answer set data is to be returned in response to a CNTQRY.

The DRDA rowset is said to be complete when the requested number of rows (*S*) are fetched or when a FETCH request at the application server results in a negative SQLSTATE or an SQLSTATE of 02000, or when the CNTQRY command identifies a positioning FETCH. The intent of a positioning FETCH is to change the position of the cursor, but explicitly does not request the return of query data. Examples are a FETCH AFTER request or a FETCH request that does not have a fetch target list. On a CNTQRY command that specifies a positioning FETCH, the DRDA rowset is considered complete upon the completion (successful or not) of the positioning FETCH request.

If the application server can only return  $R < S$  complete rows while populating the DRDA rowset before it encounters the extra query block limit, the DRDA rowset is said to be incomplete. The application server returns the *R* complete rows it has fetched as for other DRDA rowset data.

It is the application requester's responsibility to dispose of the incomplete DRDA rowset by either completing the rowset or resetting the DRDA rowset with the next CNTQRY command.

---

63. This value is an arbitrarily-chosen architectural constant that limits the number of rows returned on the OPNQRY request for a scrollable non-rowset cursor. It allows the application server to create an implicit DRDA rowset of a known size in case the application requester decides to use the cursor in a scrollable fashion with subsequent CNTQRY requests. It can be overridden by the application requester by means of the *qryrowset* parameter on the OPNQRY request for scrollable non-rowset cursors. It has no effect on subsequent CNTQRY requests if the application requester does not access the cursor in a scrollable manner (that is, does not specify a *qryrowset*).

In the case of a non-scrollable non-rowset cursor, the application requester can only receive query data sequentially. It must thus pass all the rows returned in the incomplete DRDA rowset to the application before requesting more data from the server. To request more data, the application requester must first complete the pending DRDA rowset and pass all returned rows to the application. Only after receiving all the rows in the requested DRDA rowset and returning them to the application can the application requester ask for more query data from the application server. An application requester that resets a pending DRDA rowset for a non-scrollable cursor may lose query data and thus generally will not do so.

In the case of a scrollable non-rowset cursor, the application requester can retrieve data using navigational parameters, so may either complete or reset a pending DRDA rowset to continue receiving query data without loss of query data. In either case, the application requester must first consume all data returned in the incomplete DRDA rowset, either by reading and passing all complete rows to the application or by caching or discarding any unreceived data (including EXTDTA DSSs and unreceived extra query blocks). For example, suppose the application fetches a row and then wants to fetch a row with *qryscrovn* of *qryscrel* and *qryrownbr* of *+N*. Suppose both rows are contained in the returned DRDA rowset. The first and second requested rows are consumed by being passed to the application. The rows between the two may be consumed by being discarded, including any EXTDTA DSSs data in those rows.

To complete the DRDA rowset the application requester first passes to the application all data returned (except for possibly a partial row in the last query block), then sends a CNTQRY command with a *qryrowset* parameter value of *S-R* and *qryscrovn* and *qryrownbr* parameter values equivalent to FETCH NEXT. To reset the DRDA rowset, the application requester first either passes to the application all data returned (except for possibly a partial row in the last query block), or caches or discards all data returned by the application server but not fetched by the application (including unreceived extra query blocks and unreceived EXTDTA objects sent with the reply by the application server), then sends a CNTQRY command with a *qryblkrst* value of TRUE (and possibly new *qryscrovn* and *qryrownbr* values to identify the desired row in the case of a scrollable cursor).

The application server is responsible for knowing that the last CNTQRY command resulted in an incomplete DRDA rowset and for knowing how many rows are needed to complete the DRDA rowset (*S-R*). The DRDA rowset is said to be pending at the application server. The application server validates that the next CNTQRY either completes the DRDA rowset or resets it. If the next CNTQRY command requests that the DRDA rowset be completed, then the application server returns the next rows in sequence to the application requester, up to the extra query block limit set by *maxblkext*. It is possible that the CNTQRY command may also result in an incomplete DRDA rowset, and the DRDA rowset remains pending at the server for the remaining rows. If the next CNTQRY command resets the DRDA rowset, then the application server discards any partial fetched but unsent row and any pending extra query blocks before discarding its record of the pending DRDA rowset. It prepares to return a new DRDA rowset according to the navigational and sensitivity requirements of the CNTQRY command.

**QP5** In the non-error case, the response to an EXCSQLSTT that returns result sets consists of the optional transaction component, followed by the summary component, followed by one or more query result set components.

The transaction component consists of one or more reply messages indicating the transaction state. These are ENDUOWRM, CMMRQSRM, or RDBUPDRM. If

RDBUPDRM is returned (as per the UP Rules in [Section 7.19](#) (on page 467)), it may be followed by ENDUOWRM or CMMRQSRM.

The summary component consists of a Result Set reply message (RSLSETRM), followed by an SQLCARD or SQLDTARD, followed by an SQL Result Set reply data object (SQLRSLRD). Following the SQLRSLRD reply data object, the summary component may optionally contain one or more SQLSTT reply data objects as per the setting of the *rtsetsstt* parameter on the EXCSQLSTT command. If the SQLDTARD in the summary component has any associated EXTDTAs, then the EXTDTAs are also part of the summary component but flow according to Rule CH5 (see [Section 7.22.1.3](#) (on page 478)).

Each result set component consists of the OPNQRY reply message (OPNQRYRM RPYDSS) and optional SQLCARD, followed by the optional SQL Column Information reply data object for the result set (SQLCINRD OBJDSS), followed by the FD:OCA description of the data (QRYDSC OBJDSS).

Answer set data for the result set may also be returned by the application server according to the same criteria that apply to the answer set data returned in response to the OPNQRY command, according to Rule QP1 for result sets using the Fixed Row Protocol or Rule QP2 for result sets using the Limited Block Protocol.

The application must use CNTQRY to retrieve more answer set data according to Rule QP1 for result sets using the Fixed Row Protocol or Rule QP2 for result sets using the Limited Block Protocol. The server may optionally return an RDBUPDRM reply message at the end of the reply chain (as per the UP Rules in [Section 7.19](#) (on page 467)).

### 7.22.4 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 close the cursor. Any time an implicit close occurs during processing of a cursor-related command, 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.

An SQLSTATE of 02000 may not always result in an implicit close. For example, a scrollable cursor does not get closed implicitly as a result of this SQLSTATE. For all other types of cursors, whether or not this SQLSTATE results in an implicit close depends on some other cursor properties, and the value of the *qryclsimp* parameter as specified previously on the OPNQRY command.

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, and must be the last response object for the command for that query or result set.

**QT4** The RPYDSS representing the query terminating reply message must be the first DSS in the response chain 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 7.10](#) for a description.
- When this is a reply to the DDM command GETNXTCHK.

In all other cases, the query terminating reply message RPYDSS must be chained from the OBJDSS containing the last row of answer set data, taking into account Rule CH5 (see [Section 7.22.1.3](#) (on page 478)) if it applies.

**QT5** If the normal end of the answer set is encountered and the query is one that can be closed implicitly (for example, it is non-scrollable), the ENDQRYRM reply message may be chained to an EXTDTA object returned for the query if all EXTDTAs that are to be sent to the application requester for the query have been sent and the server can change the cursor state after doing so. The ENDQRYRM and SQLCARD (optionally followed by an RDBUPDRM reply message as per the UP Rules in [Section 7.19](#) (on page 467)) are chained according to Rule CH5.

If the ENDQRYRM reply message is not returned, the query is to remain open until the next CNTQRY command is received or a CLSQRY command is received. If a CNTQRY command is received, the ENDQRYRM and SQLCARD are returned as the only replies to the command.

### 7.22.5 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 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 7-4](#) (on page 492), column H has the conditions that the cursor is open, a CNTQRY command for base row data has been received, an SQL FETCH request has returned an end-of-data SQLCA (SQLSTATE 02000). The server has determined to close the query implicitly based on the properties of the cursor and the setting of the *qryclsimp* parameter as previously sent on an OPNQRY command. The actions are to perform an SQL CLOSE cursor, mark the cursor as not open, create and place the ENDQRYRM/SQLCARD in the reply chain, and send the reply chain to the application requester.

**Note:** These tables provide additional information to clarify the behavior of the application requester or the application server. They do not exhaustively cover all cases. For example, when an SQLSTATE of '00000' is shown as being returned, this is to exemplify a successful operation. The tables do not explicitly show the possibility of a warning SQLSTATE that may also be issued in the case of successful operation, but it is a simple matter to extrapolate from the specific case given to other cases.

7.22.5.1 Rules for OPNQRY, CNTQRY, CLSQRY, and EXCSQLSTT

**Table 7-4** Application Server Rules for OPNQRY, CNTQRY, CLSQRY, EXCSQLSTT

Conditions	Cases																	
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
CURSOR STATE: NOT OPEN OPEN	A	B		D	E	F	G	H	I	J	K	L	M	N	O	P	Q	
DRDA COMMAND: CNTQRY for base row data CLSQRY OPNQRY EXCSQLSTT	A	B		D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
OPEN CURSOR FAILED																	Q	
SQL FETCH RETURNED SQLCA AND DATA ROW OR MULTI-ROWS							G											
SQL FETCH RETURNED SQLCA WITHOUT A DATA ROW QUERY TERMINATING OTHER ROLLBACK NO OTHER REPLY OBJECTS NON-QUERY TERMINATING								H	I	J	K	L						
MUST STACK RPYDSS/SQLCARD NO YES See Rules QP2, CH5, QT4, QT5								H	I									
RPYDSS/SQLCARD STACKED ENDQRYRM OTHER		B			E	F												

Actions	Cases																	
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
RETURN SQLCARD: SQLSTATE='00000' FROM RDB						F		H					M				Q	
CURSOR STATE: NOT OPEN OPEN	A	B		D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
ISSUE SQL CLOSE CURSOR								H	I				M					
CHAIN RPYDSS/SQLCARD								H										
STACK RPYDSS/SQLCARD									I									
SEND ABNUOWRM/SQLCARD											K							
STACK ABNUOWRM/SQLCARD										J								
SEND STACKED RESPONSE		B			E													
PURGE STACK		B			E	F												
SEND REPLY CHAIN						F		H	I	J		L	M					
PROCESS CNTQRY							G											
SEND QRYNOPRM RPYDSS	A			D														
PROCESS OPNQRY														N		P		
PROCESS EXCSQLSTT																		R
SEND QRYPOPRM RPYDSS															O			
SEND OPNQFLRM RPYDSS/CA																		Q

**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:

- The first stack contains the reply message ENDQRYRM. See Rules QP2, CH5, QT4, and QT5.

When an ENDQRYRM/SQLCARD is stacked on a cursor, and when the next request is a CNTQRY for additional base row data, the stacked message is sent in response to the received request. The stack is then purged. The cursor state is set to not open.

**Note:** If the next request is a CNTQRY for the externalized data associated with a previously sent base row, then the externalized data is returned as EXTDTAs and the ENDQRYRM/SQLCARD may either remain stacked or may be sent, depending on whether the server is able to change the cursor state after sending the EXTDTAs. If the server can change the cursor state and send the ENDQRYRM/SQLCARD, then the server sets the cursor state to not open; otherwise, it does not change.

- The other stack contains the reply message ABNUOWRM. This stack applies to all operations and is always the first to be processed (for all application requester requests, regardless of whether the request is for a query operation or otherwise). See Rules QT1, QT2, QT3, and QT4 in [Section 7.22.4](#) and Rule IR1.

When an ABNUOWRM/SQLCARD is stacked on a cursor, the stacked message is sent in response to 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 on the stack).

**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 7.22.3](#) (on page 484).

The SQL FETCH condition is a result of the CNTQRY process action. The CNTQRY process also includes the process of sending EXTDTAs associated with the base data in the QRYDTA objects. 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 for base data. In performing the CNTQRY process action, the application server receives a relational database query terminating response to a FETCH request, or an SQLSTATE of 02000 with a *qryclsimp* parameter previously sent on the OPNQRY command indicating that the cursor should be closed implicitly for this type of cursor. In DRDA, an implicit close is mapped to ENDQRYRM except for the rollback case which is mapped to ABNUOWRM.

CASE H represents the ENDQRYRM condition where there is no rule requiring the ENDQRYRM and SQLCARD to be stacked. The SQLCARD will contain the SQLSTATE from the relational database. The RPYDSS/SQLCARD are chained to any replies already generated for the CNTQRY command. The application server closes the cursor by requesting the relational database close the cursor. The

application server then sets the cursor state to not open and sends the reply chain.

CASE H also applies to the ABNUOWRM query terminating condition for Fixed Row Protocol for the single-row fetch case (see also CASE K).

- CASE I** This is the same as CASE H except that the RPYDSS/SQLCARD cannot be sent with the reply chain for the command. The chained replies are sent without the RPYDSS/SQLCARD. The RPYDSS/SQLCARD is stacked on the cursor waiting for the next request for base data. The cursor state remains open. Refer to Rules QT4 and QT5 in [Section 7.22.4](#) (on page 490).
- CASE J** This is the same as CASE I except the query terminating condition is a rollback which generates ABNUOWRM. Because of Rule QT4 (see [Section 7.22.4](#) (on page 490)), the ABNUOWRM must be the first DSS in the reply chain for the command. Therefore, if there are any replies already chained for the command, the ABNUOWRM/SQLCARD must be stacked waiting for the next request from the application requester. The current reply chain, with the accumulated answer set data, is sent, and the cursor state remains open.
- CASE K** This is the same as CASE J except that there are no DSSs chained as reply objects for the command. 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 for base data. In performing the CNTQRY process action, the application server receives an SQLCA without a data row in response to a FETCH request, where the error indicated 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 return the SQLCA that the relational database has provided along with a null data row and then to interrupt the process of filling the query block. The application server returns the data accumulated so far, even if there is more room in the query block for more rows, and waits for the next request. If any base rows in the interrupted query block included FD:OCA Generalized String header for externalized data, the associated EXTDTAs are returned according to Rules QP1 and QP2 in [Section 7.22.3](#) (on page 484). The application may decide to issue another FETCH, resulting in CNTQRY, to close the cursor, resulting in CLSQRY, or to rollback or terminate. The application requester is not dependent upon nor sensitive to these conditions.
- 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 if it is unable to generate a unique *qryinsid* for an instance of this query which is uniquely identified by the *cmdsrcid* as specified on the OPNQRY command. Refer to Rule SN11 in [Section 7.15](#) and Rule QI5 in [Section 7.22.2](#) for the behavior at an intermediate server.
- CASE P** 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 and is able to generate a unique *qryinsid* for an instance of this query which is uniquely identified by the *cmdsrcid* as specified on the OPNQRY command. The cursor state is set to open.
- CASE Q** 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 R** 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.

7.22.5.2 Rules for FETCH

Table 7-5 Application Requester Rules for FETCH

Conditions	Cases														
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
CURSOR STATE: NOT OPEN ONLY CLOSE ALLOWED OPEN	A	B	C	D	E	F	G	H	I						
CURRENT DSS IS QRYDTA, POSITIONED AT: SQLCA/ROW SQLCA/NULL ROW OR ROW/SQLSTATE>02999 END OF BLOCK WITH CHAINED DSS END OF BLOCK WITH NO CHAINED DSS			C	D			G	H	I						
CURRENT DSS IS RPYDSS, WITH REQUIRED SQLCARD CHAINED					E										
CURRENT DSS IS RPYDSS, WITH NO REQUIRED SQLCARD						F									
MULTI_ROW FETCH: YES NO DOESN'T MATTER:	A	B	C	D	E	F	G	H	I						

Actions	Cases														
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
RETURN SQLCA: EOQ - SQLSTATE='02000' NOT OPEN FROM AS QRYDTA FROM SQLCARD BUILT BY AR	A	B	C	D	E				I						
CURSOR STATE: NOT OPEN ONLY CLOSED ALLOWED OPEN	A	B	C	D	E		G	H	I						
RETURN ROW OR MULTI-ROWS			C						I						
ISSUE CNTQRY - RECEIVE ALL REPLY OBJECTS								H							
GET NEXT CHAINED DSS							G								
PROTOCOL ERROR						F									

**Cases for Rules**

**Note:** When comparing with the architecture in SQLAM Level 6 and below, many cases have been eliminated. To assist in comparing between different levels of the architecture, cases D, E, F, G, H, I, and J in this section correspond to cases F, I, K, L, M, N, and Q in the corresponding section of the Reference for SQLAM Level 6 and below.

**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 chosen to close the cursor implicitly based on the properties of the cursor, and the *qryclsimp* parameter as previously sent on the OPNQRY command by returning 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 a QRYDTA object on an answer set row with the associated SQLCA (may be null). It is irrelevant whether the QRYDTA is chained to a subsequent DSS.

The application requester returns the row and associated SQLCA to the application and positions itself to the next position in the QRYDTA after the row. The cursor state remains OPEN.

If the row is a base row having FD:OCA Generalized String headers, the action includes the process of retrieving the externalized data associated with each FD:OCA Generalized String header from an EXTDTA object. If the EXTDTA objects for the query are being returned a row at a time, the application requester must issue a CNTQRY 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** The application requester cursor state indicates that the cursor is open, not performing a multi-row fetch, and the application requester is positioned in a QRYDTA object on a null answer set row and a non-null SQLCA, or on a non-null answer set row where the associated SQLCA has a SQLSTATE greater than 02999. It is irrelevant whether the QRYDTA is chained to a subsequent DSS.

The application requester returns the associated SQLCA to the application and positions itself to the next position in the QRYDTA after the row. The cursor state remains open.

**CASE E** The application requester cursor state indicates that the cursor is open and the application requester is positioned at an RPYDSS which is chained to an SQLCARD.

The application requester returns an SQLCA to the application using the information in the SQLCARD. The cursor is set to the "Only CLOSE Allowed" state, meaning that the only valid operation against the cursor is to close the cursor.

**CASE F** The application requester cursor state indicates that the cursor is open and that the application requester is positioned at an RPYDSS which is not chained to a required SQLCARD. This is a protocol violation. Refer to Rule QR2.

**CASE G** The application requester cursor state indicates that the cursor is open and that the application requester is positioned at the end of a QRYDTA that is chained to a subsequent DSS.

The application requester receives the next DSS and leaves the cursor state open. Then the application requester reevaluates conditions based on the data found in the next DSS.

**CASE H** The application requester cursor state indicates that the cursor is open and that the application requester is positioned at the end of a QRYDTA that is not chained to a subsequent DSS.

The application requester issues a CNTQRY command and then receives the reply DSSs for that command. Then the application requester reevaluates conditions based on the data found in the first DSS in the reply chain.

**CASE I** The application requester cursor state indicates that the cursor is open, performing a multi-row fetch, and positioned on the first row of the SQL rowset. It is irrelevant whether the QRYDTA is chained to a subsequent DSS.

The application requester, therefore, returns a statement-level SQLCA to the application, which is derived from the SQLCARD.

**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.

If the cursor is implicitly closed after the rowset fetch, then an ENDQRYRM is returned after the EXTDTA objects (if externalized data is being returned).

7.22.5.3 Rules for CLOSE

**Table 7-6** Application Requester Rules for CLOSE

Conditions	Cases														
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
CURSOR STATE: NOT OPEN ONLY CLOSE ALLOWED OPEN	A	B	C	D	E										
RPYDSS WAS INCLUDED: YES NO			C	D	E										
RPYDSS CHAINED TO SQLCARD: YES NO			C	D											

Actions	Cases														
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
RETURN SQLCA: SQLSTATE='00000' NOT OPEN FROM AS	A	B	C		E										
CURSOR STATE: NOT OPEN	A	B	C		E										
ISSUE CLSQRY					E										
PROTOCOL ERROR				D											

**Cases for Rules**

**Note:** When comparing with the architecture in SQLAM Level 6 and below, some cases have been eliminated. To assist in comparing between different levels of the architecture, case E in this section corresponds to case F in the corresponding section of the Reference for SQLAM Level 6 and below.

**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 chosen to close the cursor implicitly based on the properties of the cursor, and the *qryclsimp* parameter as previously sent on the OPNQRY command by returning 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 included in the reply chain for the previous query command. The RPYDSS is chained to the required SQLCARD object.

The application requester returns an SQLCA to the application with an SQLSTATE of '00000', indicating that the CLOSE was successful, and sets the cursor state to not open.

**CASE D** The application requester cursor state indicates that the cursor is open. A query terminating RPYDSS is included in the reply chain for the previous query command. The RPYDSS is not chained to a subsequent DSS, and thus is not chained to the required SQLCARD object. This is a protocol violation according to Rule QT2 (see [Section 7.22.4](#) (on page 490)).

**CASE E** The application requester cursor state indicates that the cursor is open. A query terminating RPYDSS is not included in the reply chain for the previous query command. 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. Or end of data has been reached (SQLSTATE 02000), but the server has chosen not to close the cursor implicitly based on the properties of the cursor, and the *qryclsimp* parameter as previously sent on the OPNQRY command.

The application requester issues a CLSQRY command to the application server. The optional *qryclsrls* parameter can be specified to dictate whether read locks are to be freed when the query is closed. When the application requester receives a successful reply, it places the cursor in the not open state and returns the SQLCA, which is derived from the SQLCARD returned to the application requester by CLSQRY.



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 *svrcods* 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
ENDQRYRM	4,8	SQLSTATE in SQLCARD

REPLY MESSAGE	SVRCOD	SQLSTATE
ENDUOWRM	4	SQLSTATE in SQLCARD
MGRDEPRM	8	58009
MGRVLVLRM	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 SQLSTATE Codes Referenced by DRDA

- 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. Additionally, for scrollable cursors, this warning is issued when the cursor in an SQL FETCH statement was before the first row of the result table.
- 02502** The SQLSTATE reports a *No Data* exception warning due to an SQL FETCH operation that returns either an update hole or a delete hole for a scrollable sensitive static cursor.
- 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.
- 22527** Error has occurred when processing a row for a multi-row input operation.
- 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.
- 2D522** Atomic chain attempts a commit or rollback operation, either implicitly or explicitly through a commit or rollback request.

- 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.
- 56051** The application requester is unable to convert the row number value specified by the application on a FETCH request for a scrollable cursor because the row number value is larger than the largest integer that can fit into the QRYROWNBR instance variable. The FETCH request is failed.
- 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 the Data Conversion (DC Rules) DC3 to DC5 in [Section 7.7](#) (on page 440). 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.

- 560B1** The application server has failed a stored procedure call because one of the scrollable result sets returned by the stored procedure is not positioned before the first row of the result table for the cursor. This cursor position requirement ensures that the application requester can manage cursor position differences if it requests a rowset. The EXCSQLSTT command for the stored procedure call is failed and all result sets for the stored procedure call are closed.
- 560B2** The application server is unable to return an OPNQRYRM for a scrollable cursor because the application requester is not at SQLAM Level 7 or higher and thus does not support scrollable cursors. The OPNQRY command is failed and the cursor which had been successfully opened by the relational database is closed.
- 560B3** The application server is unable to return an OPNQRYRM for a scrollable result set returned by a stored procedure call because the application requester is not at SQLAM Level 7 or higher and thus does not support scrollable cursors. The EXCSQLSTT command for the stored procedure call is failed and all result sets for the stored procedure call are closed.
- 560B6** Atomic chain contains a CALL statement for a stored procedure.
- 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.  
The current SQL statement failed because the server specified does not support the requested function. The error was such that it will not preclude the successful execution of further SQL statements.
- 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.  
The current connection failed because the server does not support the requested function. A new connection is required to allow the successful execution of further

- SQL statements.
- 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 a reply message with a reply message codepoint that DRDA does not recognize or with an *srcod* 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.

**/** *Technical Standard*

**Part 2:**

**Environmental Support**

*The Open Group*



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 4.4.15.2](#) (on page 217)).

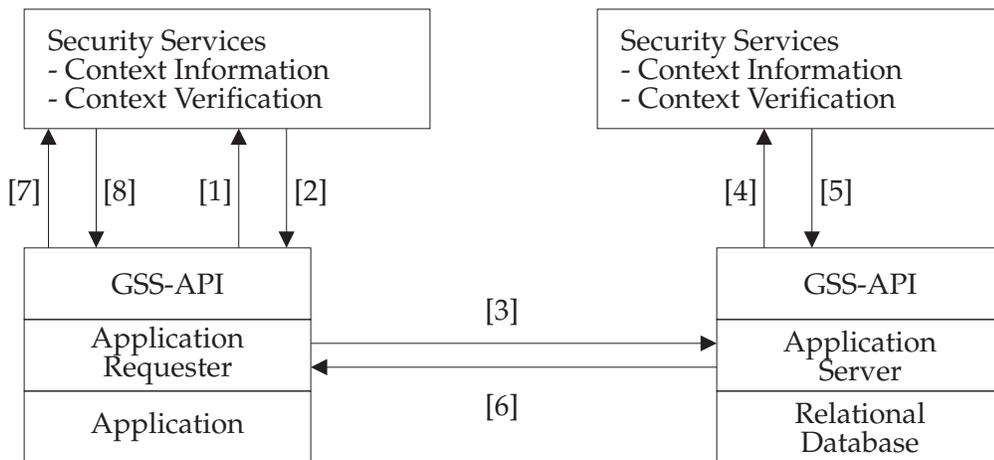
DRDA requires the ability to identify and authenticate the end user associated with the DRDA requests. Some network protocols such as LU 6.2, provide the ability to pass the information necessary to identify and authenticate the end user. See Part 3, Network Protocols for a description of this capability 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 97)). 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**. 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.

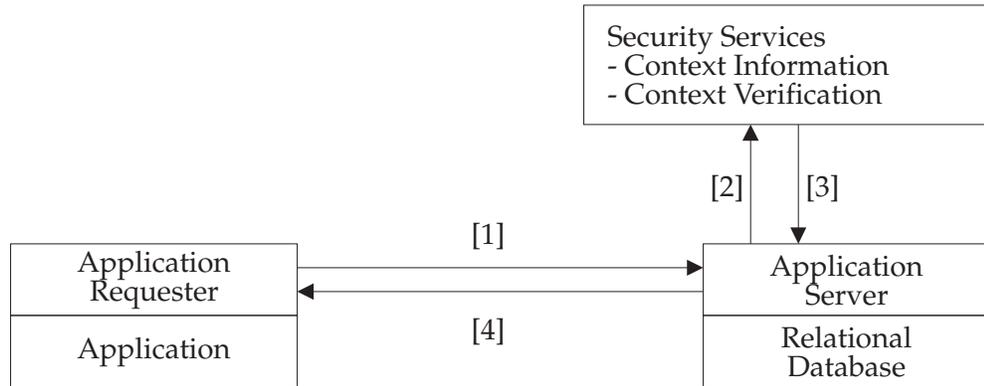
## 10.2 User ID-Related Security Mechanisms

DRDA provides the following user ID-related security mechanisms:

- User ID only
- User ID and password
- Encrypted user ID and password
- User ID and encrypted password
- User ID and password substitute
- User ID and strong password substitute
- User ID, password, and new password
- Encrypted user ID, password, and new password

The following sections provide overviews of these mechanisms.

### 10.2.1 User ID and Password

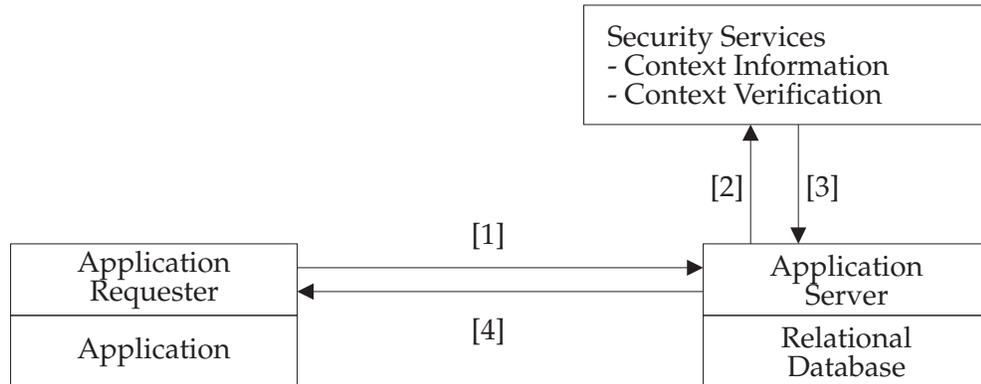


**Figure 10-2** User ID and Password Authentication

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 user ID and password and authenticate the user ID 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 user ID 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 user ID 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 User ID, Password, and New Password

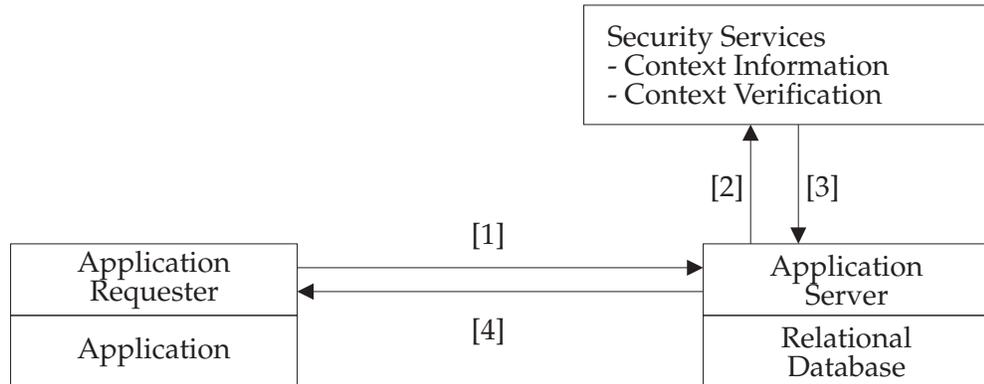


**Figure 10-3** User ID, Password, and New Password Authentication

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 user ID, password, and new password and to authenticate the user ID 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 user ID, 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 user ID 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 User ID-Only

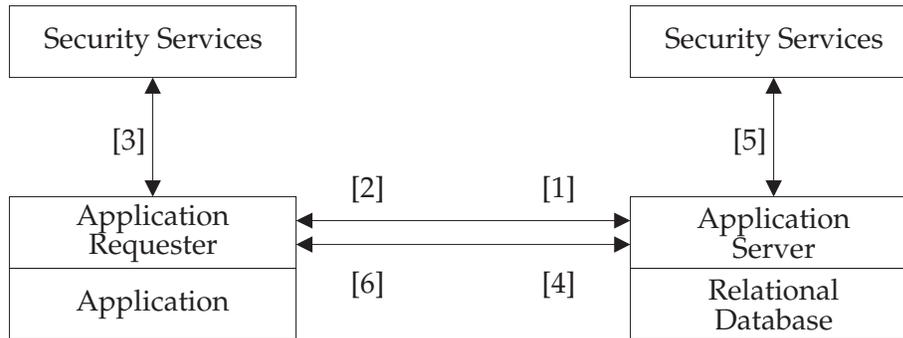


**Figure 10-4** User ID and Password Authentication

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 user ID and password and authenticate the user ID 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 user ID to the application server in the *usrid* parameter on SECCHK.
2. The application server calls the security services to process the user ID.
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 User ID and Original or Strong Password Substitute

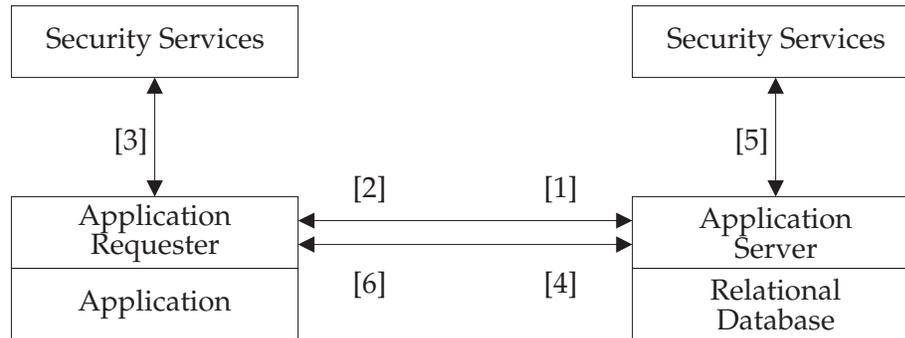


**Figure 10-5** User ID and Password Substitute Authentication

The following description of the flows applies to both the original (USRSSBPWD) and strong (USRSSBPWD) password substitute security mechanism. It 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 or USRSSBPWD and a SECTKN containing eight bytes of random data (application requester's seed). See the DHENC term in the DDM Reference for further information.
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. For the original USRSBSPWD secmec, it uses 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. For the strong USRSSBPWD secmec, it uses the Strong Hash Algorithm (SHA-1) following the procedure described in the PWDSSB term in the DDM Reference.
4. The application requester passes the user ID 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 User ID and Encrypted Password

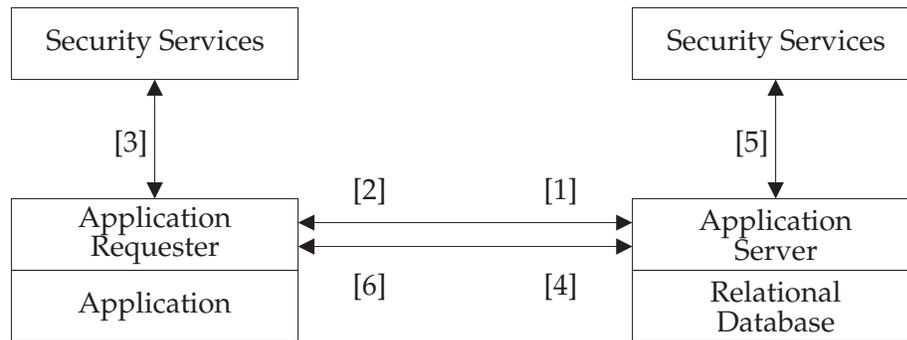


**Figure 10-6** User ID and Encrypted Password Authentication

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, user ID, and generated Diffie-Hellman shared private key described in the DDM Reference.
4. The application requester passes the user ID 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, user ID, and generated Diffie-Hellman shared private key. It then asks the local security subsystem to validate the user ID/password combination.
6. The application server returns a SECCHKRM to the application indicating success or failure based on the outcome of the authentication process.

### 10.2.6 Encrypted User ID and Password

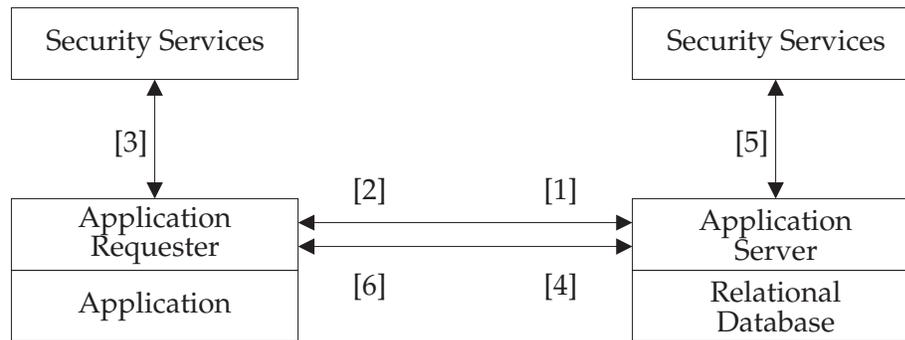


**Figure 10-7** Encrypted User ID and Password Authentication

The following description of the flows does not define the interface between the application server and the security services. It is assumed that the 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 values of EUSRIDPWD 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 DHENC 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 is also 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 user ID using the user ID, middle 8 bytes of the server's connection key (as described in the DDM Reference), and the Diffie-Hellman shared private key. The application requester then encrypts the password using the password, the middle 8 bytes of the server's connection key, and the Diffie-Hellman shared private key. Notice that the second input parameter to the DES function is the middle 8 bytes of the server's connection key.
4. The application requester passes the encrypted user ID and encrypted password to the application server in two SECTKNS included with a SECCHK.
5. The application server decrypts the encrypted user ID using the DES user ID included in the first SECTKN, the middle 8 bytes of the server's connection key, and the Diffie-Hellman shared private key. The application server then decrypts the encrypted password using the DES password included in the second SECTKN, the middle 8 bytes of the server's connection key, and the Diffie-Hellman shared private key. It then asks the local security subsystem to validate the user ID/password combination. Notice that the second input parameter to the DES function is the middle 8 bytes of the server's connection key.
6. The application server returns a SECCHKRM to the application indicating success or failure based on the outcome of the authentication process.

### 10.2.7 Encrypted User ID, Password, and New Password



**Figure 10-8** Encrypted User ID, Password, New Password Authentication

The following description of the flows does not define the interface between the application server and the security services. It is assumed that the 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 values of EUSRIDNWPWD and SECTKN contain 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 DHENC 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 is also 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 user ID using the user ID, the middle 8 bytes of the server's connection key (as described in the DDM Reference), and the Diffie-Hellman shared private key. The application requester then encrypts the password using the password, first middle 8 bytes of the server's connection key, and the Diffie-Hellman shared private key. The application requester then encrypts the new password using the password, the middle 8 bytes of the server's connection key, and the Diffie-Hellman shared private key. Notice that the second input parameter to the DES function is the middle 8 bytes of the server's connection key.
4. The application requester passes the encrypted user ID, encrypted password, and encrypted new password to the application server in three SECTKNS included with a SECCHK.
5. The application server decrypts the encrypted user ID using the DES user ID included in the first SECTKN, the middle 8 bytes of the server's connection key, and the Diffie-Hellman shared private key. The application server then decrypts the encrypted password using the DES password included in the second SECTKN, the middle 8 bytes of the server's connection key, and the Diffie-Hellman shared private key. The application server then decrypts the encrypted new password using the DES password included in the third SECTKN, the middle 8 bytes of the server's connection key, and the Diffie-Hellman shared private key. It then asks the local security subsystem to validate the user

ID/password/new password combination. Notice that the second input parameter to the DES function is the middle 8 bytes of the server's connection key.

6. The application server returns a SECCHKRM to the application indicating success or failure based on the outcome of the authentication process.

## 10.3 Kerberos

Kerberos is a security technology developed by MIT to provide users and application with secure access to data, resources, and services located anywhere on a heterogeneous network. Its purpose is to allow user authentication over a physically untrusted network. Instead of flowing user IDs and passwords “in the clear” over a network, encrypted “tickets” are used. Tickets are issued by a Kerberos authentication server. Both users and services are required to have keys registered with the authentication server. In the case of the user, this key is derived from a user-supplied password.

The environment supported by Kerberos is assumed to include an unsecure network with clients and server that are themselves not necessarily secure. Kerberos provides a means of authenticating the identity of users and authorizing access to resources independently of the security provided by the network or by the client and server systems.

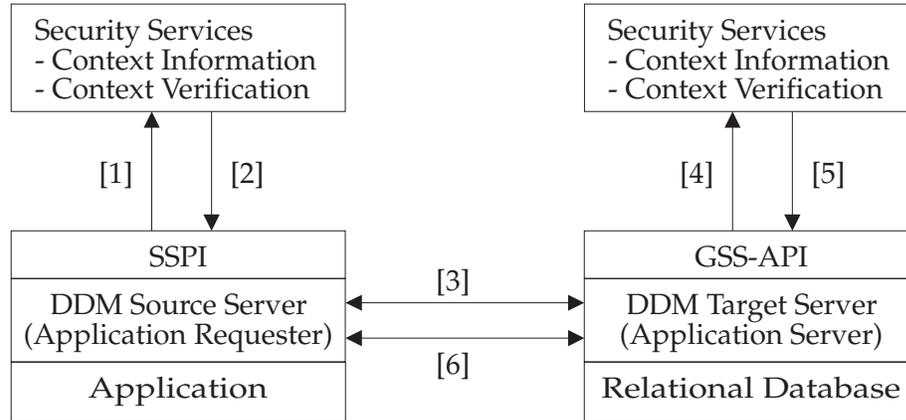
### 10.3.1 Kerberos Protocol

From a DRDA perspective, it is not necessary to understand Kerberos design and protocol. The Kerberos protocol is transparent to DRDA so it will not be described in great detail. DRDA simply provides the mechanics of flowing the Kerberos ticket. However, if users want to try and get a better understanding of Kerberos, and the underlying DES (Data Encryption Standard) encryption algorithm that is used, they can reference *Applied Cryptography—Protocols, Algorithms, and Source Code in C* by Bruce Schneier. Also, users can reference the MIT web site at <http://www.mit.edu> or <http://web.mit.edu/kerberos/www> for more information. The Kerberos protocol includes the following.

A client program logs on to Kerberos on behalf of a user (for example, UNIX users may be familiar with “kinit”). Under the covers, Kerberos acquires a Ticket-granting Ticket (TGT) from the Authentication Server. The TGT is delivered as a data packet encrypted in a secret key derived from the user’s password. Thus, only the valid user is able to enter a password and thus be able to decrypt the packet in order to obtain the use of this TGT. Whenever the client program requests services from the specific server, it must first send its TGT to the (Ticket Granting) Authentication Service to request a Ticket to access that server. The TGT enables a client program to make such requests of the authentication service and allows the authentication service to verify the validity of such a request. The ticket contains the user’s identity and information that allows the ticket to be aged, and expired tickets to be invalidated. The authentication service encrypts the ticket using a key known only to the desired server and the Kerberos Security Service. This key is known as the server key. The encrypted ticket is transmitted to the server who in turn presents it to the Kerberos authentication service to authenticate the identity of the client program and the validity of the ticket.

### 10.3.2 Kerberos Security Mechanism with SSPI and GSS-API

This section briefly describes the flows that perform Kerberos identification and authentication through the use of the Microsoft Security Support Provider Interface (SSPI) and Generic Security Services Application Programming Interface (GSS-API). An implementation may choose another interface.



**Figure 10-9** Example Kerberos-Based Flow using SSPI and GSS\_API in DRDA

The following provides a simplified overview of the flows involved with utilizing the Kerberos security mechanism. The actual Kerberos processing to perform the identification and authentication processing is described in Kerberos documentation listed in **Referenced Documents**. This example shows a source server that utilizes the SSPI-API and a target server that uses the GSS-API.

1. The application makes a request that requires access to the DDM target server. Acting on behalf of the end user of the application, the DDM source server calls the security server to obtain the security context information for accessing the DDM target server. The figure indicates a single flow, but in actuality there may be several flows. Although it is now shown in the figure, the Kerberos principal for the target DDM server, which is required for generating the security context information, may have been obtained from the target DDM server earlier by the source DDM server. Acting on behalf of the end user of the application, the source DDM server calls the security services using *AcquireCredentialsHandle()* and *InitializeSecurityContext()* to obtain the security context information for accessing the target DDM server.
2. The security server returns the security context information. The returned context information must be a Kerberos Version 5 ticket.
3. The source server passes the security context information to the target server.
4. The security service verifies the security context information. Verification is accomplished by calling the target security services using *gss\_accept\_sec\_context()* with the security context information received from the source DDM server.
5. The security service returns to the target server with an indication of the success or failure of authentication.
6. The target server returns the result, success or failure, to the source server.

### 10.3.3 Trusted Application Server

The ability for an application server to access an RDB as trusted establishes a trust relationship between the application server and the database server which can then be used to provide special privileges and capabilities to an external entity, such as a middleware server. When the database is accessed, a series of trust attributes are evaluated by the RDB to determine whether a trusted context is defined for the application server. The relationship between a connection and a trusted context is established when the connection is first created and that relationship remains for the life of that connection.

Once established, a trusted connection allows for the definition of a unique set of interactions between the database server, the application server, and the external entity including:

- Once established, the external entity can reuse an established connection under a different user ID and possibly use a different security mechanism than that used to establish the connection. Depending on the security mechanism requirements of the trusted context on the database server, it allows elimination of the need to manage end-user passwords or credentials by the external entity.
- The ability for an application server to provide a source user ID on the security check command which is different than the target user ID when the source security manager and target security manager use different user registries. In this case, a source user registry name is provided with the source user ID. If the source registry name provided is different than the target registry name used by the database server, the target security manager must have an association defined between the source and target registries. If an association is defined, the target security manager uses the source user ID, the source registry name, and the security token to authenticate the user and then maps the source user ID to the target user ID. The target user ID is provided to the RDB to establish the database authorization ID.
- The ability for an application server to use additional privileges within a trusted connection that are available to it outside the trusted connection. This can be accomplished by the RDB associating special privileges for the external entity when accessed as trusted.

If the connection is established using an encryption security mechanism, the shared private key is used for the life of the connection and is not established when the user ID associated with the connection changes.

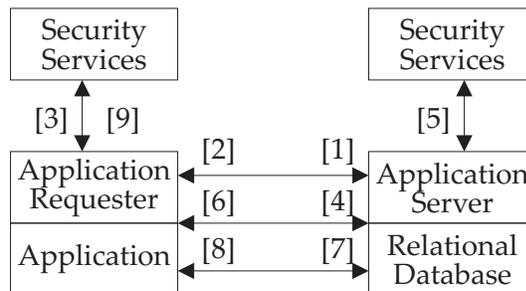
## 10.4 User ID and Data-Related Security Mechanisms

DRDA provides the following user ID and data-related security mechanisms:

- Encrypted user ID and security-sensitive data
- Encrypted user ID, password, and security-sensitive data
- Encrypted user ID, password, new password, and security-sensitive data

The following section provides an overview of this mechanism.

### 10.4.1 Encrypted User ID and Security-Sensitive Data



**Figure 10-10** Encrypted User ID and Security-Sensitive Data

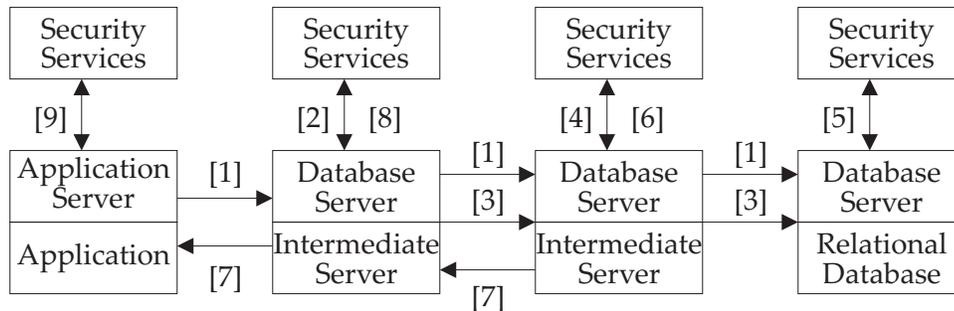
The following description of the flows does not define the interface between the application server and the security services. It is assumed that the local services are available at the application requester and at the application server to perform the required functions described below.

1. The application makes a request that requires access to the application server. The application requester sends an ACCSEC command with a SECMEC value of EUSRIDDTA and SECTKN containing the application requester's connection key, which is generated using the standard Diffie-Hellman key distribution algorithm. In this flow, the application requester uses default values for the ENCALG and ENCKEYLEN parameters and thus does not include them in the ACCSEC command. See the DHENC term in the DDM Reference.
2. The application server saves the requester's connection key and generates the server's connection key and shared private key using the Diffie-Hellman algorithm. It then derives the encryption seed from the shared private key and the encryption token from the server's connection key. The application server replies with ACCSECRD containing the server's connection key in SECTKN.
3. The application requester saves the server's connection key and generates the shared private key using the Diffie-Hellman algorithm. It then derives the encryption seed from the shared private key and the encryption token from the server's connection key. The application requester encrypts the user ID using the user ID, the encryption token, and the encryption seed.
4. The application requester sends the encrypted user ID to the application server in a SECTKN included with a SECCHK.
5. The application server decrypts the encrypted user ID using the user ID included in the SECTKN, the encryption token, and the encryption seed. It then asks the local security subsystem to validate the user ID.

6. The application server returns a SECCHKRM to the application indicating success or failure based on the outcome of the authentication process.
7. On successful authentication, for security-sensitive data that accompanies any DRDA command, the application requester encrypts the DSS carrier containing the security-sensitive data, using the given data, the encryption token, and the encryption seed. The application requester sets *dsstype* in the DSS header to indicate that the object encapsulated in the DSS is encrypted. The application requester sends the encrypted data to the server.
8. The application server decrypts the encrypted data using the data, the encryption token, and the encryption seed. Similarly, for any security-sensitive data, the application server encrypts the DSS carrier containing the security-sensitive data, using the given data, the encryption token, and the encryption seed. The application server sets the *dsstype* in the DSS header to indicate that the object encapsulated in the DSS is encrypted and sends the encrypted data to the requester.
9. The application requester decrypts the encrypted data using the encrypted data, the encryption token, and the encryption seed.

### Intermediate Server Processing

This section briefly describes the intermediate server processing for the Encrypted User ID and Security-sensitive Data Security Mechanism. This section describes the intermediate server processing for security-sensitive objects using SECTKNOVR. Alternatively, the intermediate server can always decrypt and re-encrypt the security-sensitive data.



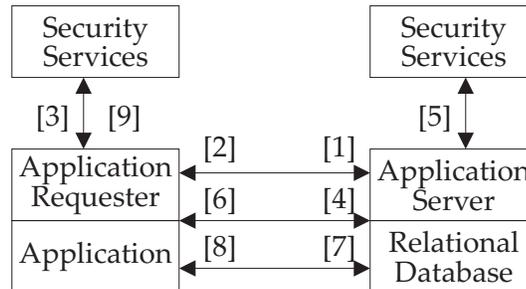
**Figure 10-11** Intermediate Server Encrypted User ID and Security-Sensitive Data

The following description of the flows does not define the interface between the intermediate server and the security services. It is assumed that the local services are available at the application requester, at the intermediate server and at the application server to perform the required functions described below.

1. During ACCSEC/ACCSECRD processing, the intermediate server negotiates the EUSRIDDTA security mechanism with the downstream site.
2. On successful connection, if the request data stream contains encrypted security-sensitive objects, then the intermediate server passes the encryption seed and the encryption token used to encrypt the security-sensitive data in SECTKNOVR, encryption seed SECTKN first and encryption token SECTKN second. The SECTKNOVR DSS is encrypted using the encryption token and the encryption seed exchanged with the downstream site.
3. The intermediate server sends the encrypted SECTKNOVR, with the encrypted security-sensitive objects to the downstream site.
4. If the intermediate server receives an encrypted SECTKNOVR along with encrypted security-sensitive objects, the intermediate server decrypts the SECTKNOVR using the encryption token and the encryption seed exchanged with the upstream site. The intermediate server then re-encrypts the SECTKNOVR using the encryption token and the encryption seed exchanged with the downstream site.
5. The database server first decrypts the encrypted SECTKNOVR using the encryption seed and the encryption token. The database server then decrypts the encrypted security-sensitive objects using the decrypted encryption seed in the first SECTKN and the decrypted encryption token in the second SECTKN of SECTKNOVR.
6. If the reply data stream contains encrypted security-sensitive objects, then the intermediate server passes the encryption seed and the encryption token used to encrypt the security-sensitive objects in SECTKNOVR, encryption seed SECTKN first and encryption token SECTKN second. The SECTKNOVR DSS is encrypted using the encryption token and the encryption seed exchanged with the upstream site.
7. The intermediate server sends the encrypted SECTKNOVR, with the encrypted security-sensitive objects to the upstream site.

8. If the intermediate server receives an encrypted SECTKNOVR along with encrypted security-sensitive objects, the intermediate server decrypts the SECTKNOVR using the encryption token and the encryption seed exchanged with the downstream site. The intermediate server then re-encrypts the SECTKNOVR using the encryption token and the encryption seed exchanged with the upstream site.
9. The application server first decrypts the SECTKNOVR using the encryption token and the encryption seed. The application server then decrypts the encrypted security-sensitive objects using the decrypted encryption seed in the first SECTKN and the decrypted encryption token in the second SECTKN of SECTKNOVR.

### 10.4.2 Encrypted User ID, Password, and Security-Sensitive Data



**Figure 10-12** Encrypted User ID, Password, and Security-Sensitive Data

The following description of the flows does not define the interface between the application server and the security services. It is assumed that the local services are available at the application requester and at the application server to perform the required functions described below.

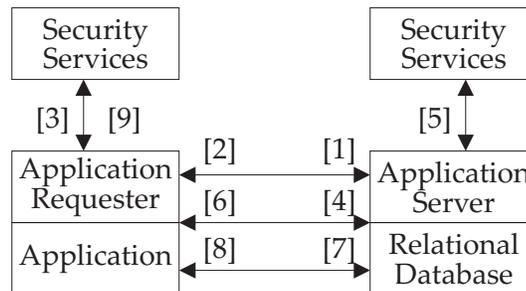
1. The application makes a request that requires access to the application server. The application requester sends an ACCSEC command with a SECMEC value of EUSRPWDDTA and SECTKN containing the application requester's connection key, which is generated using the standard Diffie-Hellman key distribution algorithm. In this flow, the application requester uses default values for ENCALG and ENCKEYLEN parameters and thus does not include them in the ACCSEC command. See the DHENC term in the DDM Reference.
2. The application server saves the requester's connection key and generates the server's connection key and shared private key using the Diffie-Hellman algorithm. It then derives the encryption seed from the shared private key and the encryption token from the server's connection key. The application server replies with ACCSECRD containing the server's connection key in SECTKN.
3. The application requester saves the server's connection key and generates the shared private key using the Diffie-Hellman algorithm. It then derives the encryption seed from the shared private key and the encryption token from the server's connection key. 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 encrypts the user ID using the user ID, the encryption token, and the encryption seed. The application requester then encrypts the password using the password, the encryption token, and the encryption seed.
4. The application requester sends the encrypted user ID and encrypted password to the application server in two SECTKNs included with a SECCHK.
5. The application server decrypts the encrypted user ID using the user ID included in the first SECTKN, the encryption token, and the encryption seed. The application server then decrypts the encrypted password using the password included in the second SECTKN, the encryption token, and the encryption seed. It then asks the local security subsystem to validate the user ID/password combination.
6. The application server returns a SECCHKRM to the application indicating success or failure based on the outcome of the authentication process.

7. On successful authentication, for any security-sensitive data that accompanies any DRDA command, the application requester encrypts the DSS carrier containing the security-sensitive data, using the given data, the encryption token, and the encryption seed. The application requester sets the *dsstype* in the DSS header to indicate that the object encapsulated in the DSS is encrypted. The application requester sends the encrypted data to the server.
8. The application server decrypts the encrypted data using the data, the encryption token, and the encryption seed. Similarly, for any security-sensitive data, the application server encrypts the DSS carrier containing the security-sensitive data, using the given data, the encryption token, and the encryption seed. The application server sets the *dsstype* in the DSS header to indicate that the object encapsulated in the DSS is encrypted and sends the encrypted data to the requester.
9. The application requester decrypts the encrypted data using the encrypted data, the encryption token, and the encryption seed.

#### **Intermediate Server Processing**

The intermediate server processing flows for Encrypted User ID, Password, and Security-sensitive data are identical to those of Encrypted User ID and Security-sensitive data intermediate server processing flows, except that during ACCSEC/ACCSECRD processing the intermediate server negotiates the EUSRPWDDTA security mechanism with the downstream site.

### 10.4.3 Encrypted User ID, Password, New Password, and Security-Sensitive Data



**Figure 10-13** Encrypted User ID, Password, New Password, and Security-Sensitive Data

The following description of the flows does not define the interface between the application server and the security services. It is assumed that the 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 a SECMEC value of EUSRNPWDDTA and SECTKN containing the application requester's connection key, which is generated using the standard Diffie-Hellman key distribution algorithm. In this flow, the application requester uses default values for ENCALG and ENCKEYLEN parameters and thus does not include them in the ACCSEC command. See the DHENC term in the DDM Reference.
2. The application server saves the requester's connection key and generates the server's connection key and shared private key using the Diffie-Hellman algorithm. It then derives the encryption seed from the shared private key and the encryption token from the server's connection key. The application server replies with ACCSECRD containing the server's connection key in SECTKN.
3. The application requester saves the server's connection key and generates the shared private key using the Diffie-Hellman algorithm. It then derives the encryption seed from the shared private key and the encryption token from the server's connection key. 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 encrypts the user ID using the user ID, the encryption token, and the encryption seed. The application requester encrypts the password using the password, the encryption token, and the encryption seed. The application requester then encrypts the new password using the new password, the encryption token, and the encryption seed.
4. The application requester sends the encrypted user ID, encrypted password, and encrypted new password to the application server in three SECTKNs included with a SECCHK.
5. The application server decrypts the encrypted user ID using the user ID included in the first SECTKN, the encryption token, and the encryption seed. The application server decrypts the encrypted password using the password included in the second SECTKN, the encryption token, and the encryption seed. The application server then decrypts the encrypted new password using the new password included in the third SECTKN, the encryption token, and the encryption seed. It then asks the local security subsystem to validate the user ID/password/new password combination.

6. The application server returns a SECCHKRM to the application indicating success or failure based on the outcome of the authentication process.
7. On successful authentication, for any security-sensitive data that accompanies any DRDA command, the application requester encrypts the DSS carrier containing the security-sensitive data, using the given data, the encryption token, and the encryption seed. The application requester sets the *dsstype* in the DSS header to indicate that the object encapsulated in the DSS is encrypted. The application requester sends the encrypted data to the server.
8. The application server decrypts the encrypted data using the data, the encryption token, and the encryption seed. Similarly, for any security-sensitive data, the application server encrypts the DSS carrier containing the security-sensitive data, using the given data, the encryption token, and the encryption seed. The application server sets the *dsstype* in the DSS header to indicate that the object encapsulated in the DSS is encrypted and sends the encrypted data to the requester.
9. The application requester decrypts the encrypted data using the encrypted data, the encryption token, and the encryption seed.

#### **Intermediate Server Processing**

The intermediate server processing flows for Encrypted User ID, Password, New Password, and Security-sensitive data are identical to those of Encrypted User ID and Security-sensitive data intermediate server processing flows, except during ACCSEC/ACCSECRD processing the intermediate server negotiates the EUSRNPWDDTA security mechanism with the downstream site.

### 10.5 Plug-In Security Mechanism

A security plug-in is a module that may be inserted into the security service used by a server for the purpose of providing the context information and performing the context verification. This has the advantage of being able to use multiple underlying security mechanisms through a common interface such as GSS-API. As a result, the number of security mechanisms available for use can readily and easily be extended from the presently defined DRDA security mechanisms.

DRDA has no requirement to understand the operation of the security plug-in; only that it must facilitate the negotiation of the plug-in to be used and that the plug-in will provide the context information in mechanism-specific tokens. These tokens are opaque data from the DRDA perspective. Obviously, the plug-ins at the target and source must be compatible to understand and process the tokens. Compatible plug-ins will be assumed to have identical plug-in names.

Figure 10-14 provides a simplified overview of the flows involved in using the Plug-in security mechanism. Following the figure is a description of the flows.

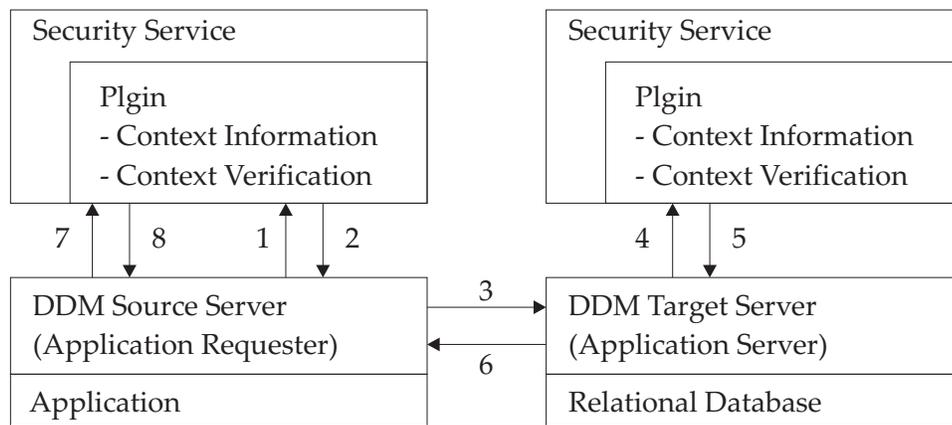


Figure 10-14 Example of Plug-In-Based Flows

The following is a brief description of the example plug-in flows shown in Figure 10-14 (on page 536).

1. The DDM source server calls the security service to obtain the security context information for accessing the DDM target server. The figure indicates a single flow, but in actuality there may be several flows.
2. The security service returns the security context information. It will also indicate whether it expects additional information from the target server. Note that security context information may be returned even though the security service indicates an error.
3. The source server passes the security context information to the target server.
4. The security service verifies the security context information via the plug-in module.
5. The security service returns to the target server with an indication of the success or failure of authentication. In addition, security context information destined for the source server may be returned to the DDM target server; often this is mutual authentication information. Note that this information may be returned even though the security service indicates an error.

6. The target server returns the result, success or failure, to the source server. The security context information, if present, will also be sent.
7. The source server calls the security service to verify the security context information received from the target server, if present.
8. The security service returns the results to the source server.



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**.

## 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 for SNA and IPv4:

```
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.

For IPv6-related correlation values, the *x* component of the CRRTKN is being extended from "1 to 8 bytes" to "1 to 39 bytes". The *z* component of the CRRTKN is being extended from "1 to 6" bytes to "1 to 12" bytes (character). An additional dot is being added to separate the *y* and *z* component. The changes for IPv6 are as follows:

Generic CRRTKN format for IPv6:

```
x.y.z  where x is 1 to 39 bytes (variable), character
        y is 1 to 8 bytes (variable), character
        z is 12 bytes (fixed), character
with a period (".") to delimit x from y, and a period to
delimit y from z, the total byte count is a variable
between 16 and 61.
```

**Note:** The generic CRRTKN format for IPv6 is exactly the same as the displayable format; see [Section 11.3.1](#) (on page 543).

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](#) and [Section 13.6.1](#) 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.

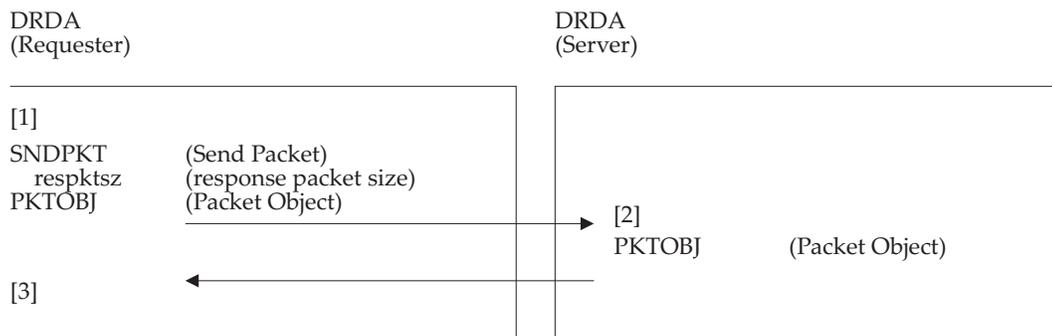
## 11.2 Monitoring

### 11.2.1 Verification of Network Connectivity Flag

To test connectivity to an application server or database server, a requester can send a trivial EXCSAT command (one containing no parameters) and listen for its associated reply containing an EXCSATRD reply data object. Network response time can also be measured, especially when this flow is repeated multiple times. The requester can use this flow to implement a tool similar to the TCP/IP *ping* utility.

### 11.2.2 Request and Response Packet Object

The SNDPKT command could also be used to test the connectivity between requester and server. It can be used to check whether the connection is healthy or not. By using the varying size of packet objects and timing the SNDPKT command to be executed, SNDPKT will provide the status of the connection and network. Following is a command flow on the SNDPKT command:



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 command can flow over a DRDA connection at any time after the DDM EXCSAT command.  
 Send a packet from Requester to Server.  
 The requester specifies the response packet size to be returned from the server.  
 The packet object will also be sent to the server; the content of it will be ignored.
2. The server receives the SNDPKT command and builds a reply packet object by using the *respktsz* value. The server sends the PKTOBJ reply packet object to the requester.
3. The requester receives the reply packet object.

### 11.2.3 Elapsed Time

A requester can request a server to monitor an event or an activity. This allows the administrator at the client the ability to obtain realtime server performance and statistical information. This information is returned to the client and can be used by the administrator to perform problem determination. Monitoring is specified on a per-request basis. Each request sent to the target server can request monitoring data to be returned as reply data in the reply. Monitoring data is not returned as a new reply message but as additional data to the reply. If monitoring fails or a specific type of monitoring is not supported, the request should not fail. The infrastructure is defined to allow an application requester to request the application server to monitor any number of events or activities while processing the request and optionally return monitoring data to the client for analysis. Currently, DRDA has defined only one activity that can be monitored. The requester can request the server provide the elapsed time required to parse the request data stream, process the request, and generate the reply data stream. Network time for sending or receiving is not included in the elapsed time. Yet as requirements are identified, new monitoring items can be added to the existing monitoring model.

The monitor instance variable can be specified on a command. The instance variable contains a list of activities or events to be monitored by the server. If monitoring is requested, an optional MONITORRD reply data is returned in the reply chain containing the requested data. For example:

```
AR Command: EXCSQLIMM (MONITOR(ETIME))
AS Reply: ...SQLCARD, .MONITORRD
AR Command: .OPNQRY (MONITOR(ETIME))
AS Reply: ...OPNQRYRM, .QRYDSC, QRYDTA, .MONITORRD
```

This support allows monitoring information to be processed independently of normal command processing. The cost of verifying the monitoring instance variables and generating the monitor reply data occurs only if monitoring is requested.

### 11.2.4 Ping

To monitor the connectivity to an application server or database server, a DRDA *ping* command can be defined using the exchange server attributes (EXCSAT) command. The EXCSAT command verifies a connection to a remote RDB, by sending an EXCSAT request to the DRDA server and listening for the EXCSATRD reply. The *ping* command waits for up to 1 second for each EXCSAT sent and prints the number of EXCSATs transmitted and EXCSATRD received. Each received EXCSATRD is validated against the transmitted EXCSAT request. Since DRDA allows the EXCSAT to be sent multiple times, no explicit changes to the architecture are needed to support a DRDA *ping* command.

## 11.3 DRDA Required Problem Determination and Isolation Enhancements

This section describes the DRDA requirements regarding correlation displays and collecting diagnostic information.

### 11.3.1 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 for SNA and IPv4 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

2. For IPv6, the generic CRRTKN format is the displayable format. The *x* component represents the readable IPV6 address, colon seperated. The *y* component represents the readable port value.

Displaying a CRRTKN value:

x.y.z where x is 1 to 39 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,  
 the total byte count is a variable between 16 and 61.

TCP/IP example:

1111:2222:3333:4444:5555:6666:7777:8888.65535.0123456789AB

### 11.3.2 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

#### 11.3.2.1 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 a reply message listed in [Table 11-1](#) (on page 547). 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.

11.3.2.2 *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.

## 11.4 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.

### 11.4.1 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 11.4.3](#) 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.

### 11.4.2 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 codepoints required.

The references listed in **Referenced Documents** should be used to gain a more complete understanding of the architecture of generic alerts.

#### 11.4.2.1 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](#) categorizes the subvectors, subfields, and codepoints 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

### 11.4.3 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.

#### 11.4.3.1 Specific Alert to DDM Reply Message Mapping

Table 11-1 defines the alert constructs to DDM reply messages. The numbers following the reply message in column one are the severity codes (*svrcods*) of the reply messages. The column labeled *where* is the location in which the alert is to be generated. For each DDM reply message created at the application server, the specified alert must be generated at the application server. For each DDM reply message 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 <a href="#">Table 11-3</a> (on page 549))	See alert model and the DDM Reference for information on DDM reply message AGNPRMRM.
CMDCHKRM 8,16,32,64,128	AR/AS	CMDCHK (see <a href="#">Table 11-6</a> (on page 555))	See alert model and the DDM Reference for information on DDM reply message CMDCHKRM.
CMDVLTRM 8	AR/AS	CMDVLT (see <a href="#">Table 11-7</a> (on page 556))	See alert model and the DDM Reference for information on DDM reply message CMDVLTRM.
DSCINVRM 8	AR/AS	DSCERR (see <a href="#">Table 11-8</a> (on page 557))	See alert model and the DDM Reference for information on DDM reply message DSCINVRM.
DTAMCHRM 8	AR/AS	DSCERR (see <a href="#">Table 11-8</a> (on page 557))	See alert model and the DDM Reference for information on DDM reply message DTAMCHRM.
PRCCNVRM 8,16,128	AR/AS	PRCCNV (see <a href="#">Table 11-10</a> (on page 559))	See alert model and the DDM Reference for information on DDM reply message PRCCNVRM.
QRYNOPRM 8	AR/AS	QRYERR (see <a href="#">Table 11-11</a> (on page 560))	See alert model and the DDM Reference for information on DDM reply message QRYNOPRM.
QRYPOPRM 8	AR/AS	QRYERR (see <a href="#">Table 11-11</a> (on page 560))	See alert model and the DDM Reference for information on DDM reply message QRYPOPRM.
RDBNACRM 8	AR/AS	RDBERR (see <a href="#">Table 11-12</a> (on page 561))	See alert model and the DDM Reference for information on DDM reply message RDBNACRM.
RDBACCRM 8	AR/AS	RDBERR (see <a href="#">Table 11-12</a> (on page 561))	See alert model and the DDM Reference for information on DDM reply message RDBACCRM.
RSCLMTRM 16,32,64,128	AS	RSCLMT (see <a href="#">Table 11-13</a> (on page 562))	See alert model and the DDM Reference for information on DDM reply message RSCLMTRM.
RSCLMTRM 8	AS	RSCLMT (see <a href="#">Table 11-13</a> (on page 562))	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.
SECCHKRM 16	AR/AS	SECVIOL (see <a href="#">Table 11-14</a> (on page 564))	See alert model and the DDM Reference for information on DDM reply message SECCHKRM.

DDM RM	Where	Alert Model	Additional Information
SYNTAXRM 8	AR/AS	SYNTAX (see <a href="#">Table 11-15</a> (on page 565))	See alert model and the DDM Reference for information on DDM reply message SYNTAXRM.

#### 11.4.3.2 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 reply message, 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 <a href="#">Table 11-13</a> (on page 562))	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 <a href="#">Table 11-4</a> (on page 553))	See Block Formats (BF Rules) in <a href="#">Section 7.22.1.1</a> (on page 474).
Chaining Violation	CHNVIO (see <a href="#">Table 11-5</a> (on page 554))	See Chaining (CH Rules) in <a href="#">Section 7.22.1.3</a> (on page 478).
DEALLOCATE type ABEND received from the application server without an accompanying DDM reply message from the application server	GENERR (see <a href="#">Table 11-9</a> (on page 558))	See <i>SNA Transaction Programmer's Reference Manual for LU Type 6.2</i> (GC30-3084, IBM) for more information on DEALLOCATE ABEND.
DSS error: Error in the Data Stream Structure received from the application server	SYNTAX (see <a href="#">Table 11-15</a> (on page 565))	See the DDM Reference for more information on Data Stream Structures.

#### 11.4.3.3 DRDA-Defined Alert Models

The next series of tables are models of alert categories DRDA uses. [Table 11-1](#) 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](#) is a description of the subvectors, subfields, and codepoints. Because the majority of the subvectors, subfields, and codepoints are common, the subsequent tables reference [Table 11-3](#) 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**.

### 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 codepoint X'F0A3' should contain the following data beginning at byte 7. The subfield X'85' uses the data ID codepoint X'0087' for relational database. The detailed data field contains the actual RDB\_NAME of the target relational database.

sf85

rdbname
---------

7

**Figure 11-2** Subfield X'85' for Failure Causes Codepoint X'F0A3'

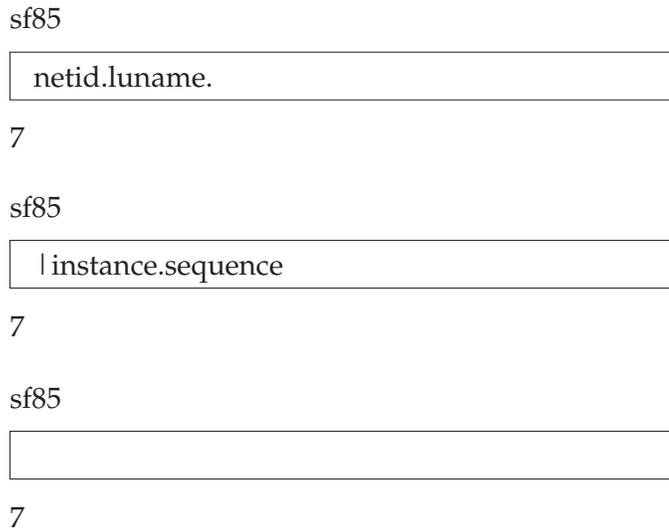
### Actions

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 codepoints define the recommended actions. The implementing products should choose the codepoints that best fit their environment. Those codepoints should be listed in the order of priorities with the most important coming first followed by the next most important.

Action codepoint X'32D0' should be used to report symptom string information if it is available.

DRDA requires codepoint X'32D1'. This codepoint 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 603). The three subfield X'85's in this codepoint

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 (on page 551), the first subfield X'85' contains the NETID.LUNAME portion of the SNA LUWID followed by a period. In the case of TCP/IP connections, the first subfield X'85' should contain the x.y netname portion of the UOWID, followed by a period. In the case of TCP/IP IPv4 connections, the x.y portion of the UOWID is typically, but not necessarily, an IPADDR.PORT. In the case of TCP/IP IPv6 connections, the IPADDR cannot be represented so another unique value is substituted for the x.y netname portion. 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.



**Figure 11-3** Subfield X'85's for Actions Codepoint X'32D1'

DRDA Level 2 requires codepoint X'32A0' if the *crrtkn* is available. This codepoint 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 codepoint of X'0101' for correlation ID. Figure 11-4 displays the subfield X'85' that is associated with this codepoint.



**Figure 11-4** Subfield X'85' for Actions Codepoint 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 codepoint for resource type identifier should be X'42' for requester.
2. *AS* to represent application server. The codepoint 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](#) for a description of the subvectors, subfields, and codepoints.

**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](#) for a description of the subvectors, subfields, and codepoints.

**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](#) for a description of the subvectors, subfields, and codepoints.

**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](#) for a description of the subvectors, subfields, and codepoints.

**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](#) for a description of the subvectors, subfields, and codepoints.

**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](#) for a description of the subvectors, subfields, and codepoints.

**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](#) for a description of the subvectors, subfields, and codepoints.

**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](#) for a description of the subvectors, subfields, and codepoints.

**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](#) for a description of the subvectors, subfields, and codepoints.

**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**

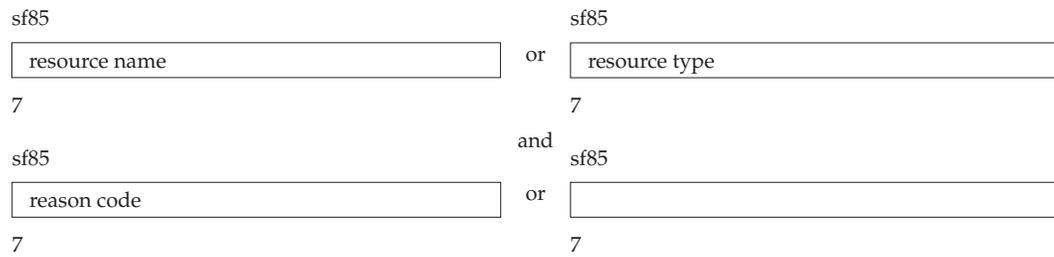
This alert model is for the Resource Limit Reached condition. See the description for [Table 11-3](#) for a description of the subvectors, subfields, and codepoints.

**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 codepoint X'F0C0' should contain the following data beginning at byte 5 (see [Figure 11-5](#) (on page 563)). 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 codepoint 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

codepoint for the resource type should be used. The second subfield X'85' uses the data ID codepoint 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 codepoint of X'0000' and does not contain any data.



**Figure 11-5** Subfield X'85's for Failure Causes Codepoint 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](#) for a description of the subvectors, subfields, and codepoints.

**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](#) for a description of the subvectors, subfields, and codepoints.

**Table 11-15** Alert Model SYNTAX

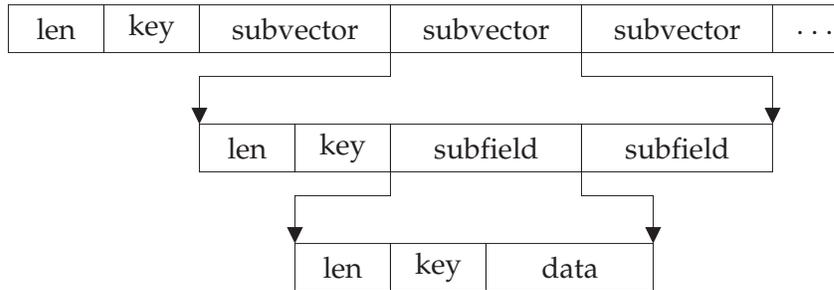
Alert ID Number		X'C299284E'
Alert Type	X'01'	Permanent
Alert Description	X'2102'	Distributed Process Failed
Probable Causes	X'1054'	Invalid Data Structure
User Causes	(none)	
Install Causes	(none)	
Failure Causes	X'1054' X'F0A3'	Invalid Data Structure Error Failure Occurred On (sf85)
Actions	X'32D1'	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)
	...	...

### 11.4.4 Alert Example

This section provides an alert example.

#### 11.4.4.1 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 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 codepoints (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 codepoint (2 bytes).
- 51 Length of Failure Causes Subfield in binary (1 byte).
- 52 Key for Failure Causes Subfield (1 byte).
- 53 Failure Causes codepoint (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 codepoint 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 codepoint indicating Review Supporting Data at Alert Sender (2 bytes).

<b>A7</b>	Length of Product Set ID Subvector in binary (1 byte).
<b>A8</b>	Key for Product Set ID Subvector (1 byte).
<b>A9</b>	Retired (1 byte).
<b>AA</b>	Length of Product Identifier Subvector in binary (1 byte).
<b>AB</b>	Key for Product Identifier Subvector (1 byte).
<b>AC</b>	Bits 0-3 are reserved. Bits 4-7 equal 4 to indicate the software level (1 byte).
<b>AD</b>	Length of Product Identifier Subfield in binary (1 byte).
<b>AE</b>	Key for Product Identifier Subfield (1 byte).
<b>AF</b>	Software product program number. Seven uppercase alphanumeric EBCDIC characters (7 bytes).
<b>B6-C8</b>	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).
<b>CA</b>	Length of Date/Time Subvector in binary (1 byte).
<b>CB</b>	Key for Date/Time Subvector (1 byte).
<b>CC</b>	Indicates the Date/Time is the local Date/Time (1 byte).
<b>CD</b>	Length of Local Date/Time Subfield in binary (1 byte).
<b>CE</b>	Key for Local Date/Time Subfield (1 byte).
<b>CF-D1</b>	The year, month, and day in binary (3 bytes).
<b>D2-D4</b>	The hours, minutes, and seconds in binary (3 bytes).
<b>D5</b>	The extension of time in binary and provides fractions of seconds (2 bytes).
<b>D7</b>	Length of Supporting Data Correlation Subvector in binary (1 byte).
<b>D8</b>	Key for Supporting Data Correlation Subvector (1 byte).
<b>D9</b>	Length of Detailed Data Subfield in binary (1 byte).
<b>DA</b>	Key for Detailed Data Subfield (1 byte).
<b>DB</b>	Product ID code (1 byte).
<b>DC</b>	Reserved (1 byte).
<b>DD</b>	Data ID equals X'00DA' for Log ID (2 bytes).
<b>DF</b>	Data encoding equals 11 for character set 00640-0500 (1 byte).
<b>E0</b>	Detailed data. The example shows Sys1.Logrec as the log ID (11 bytes).

**/** *Technical Standard*

**Part 3:**

**Network Protocols**

*The Open Group*



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**.

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** 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)<sup>64</sup>  
ALLOCATE
- Program-Supplied User ID and Password (Conversation-Level Security) (213)<sup>65</sup>  
ALLOCATE
- Specify a synchronization level of SYNCPT (108)<sup>66</sup>  
ALLOCATE
- Get the conversation state (108)<sup>67</sup>  
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 base set function provided by the following type-independent conversation verb:

- LUW\_Identifier (243)  
GET\_TP\_PROPERTIES
- Protected\_LUW\_Identifier (108)<sup>68</sup>  
GET\_TP\_PROPERTIES

---

64. LU 6.2 Conversation-Level Security is optional if DCE user authentication mechanisms are in use.

65. LU 6.2 Conversation-Level Security is optional if DCE user authentication mechanisms are in use.

66. Not supported in DRDA Level 1.

67. Not supported in DRDA Level 1.

68. Not supported in DRDA Level 1.

- SYNCPT (108)<sup>69</sup>
- BACKOUT (108)<sup>70</sup>
- SET\_SYNCPT\_OPTIONS (108)<sup>71</sup>

---

69. 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.

70. 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.

71. 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](#) and [Section 6.1.1](#) for 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](#) 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](#) and [Section 6.1.1](#) 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](#) 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,<sup>72</sup> 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<sup>73</sup> 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.

#### **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

72. 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.

73. For protected conversations in DRDA Level 2

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 and Figure 12-3 depicts DDM command processing using the LU 6.2 synchronous wait protocol verbs. DRDA also permits asynchronous wait protocols. Figure 12-1 depicts the initialization flows while using LU 6.2 security. Figure 12-3 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,<sup>74</sup> 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 for further detail about DRDA DDM command sequences.

The DRDA initialization flow while using LU 6.2 security consists of the following:

---

74. 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.

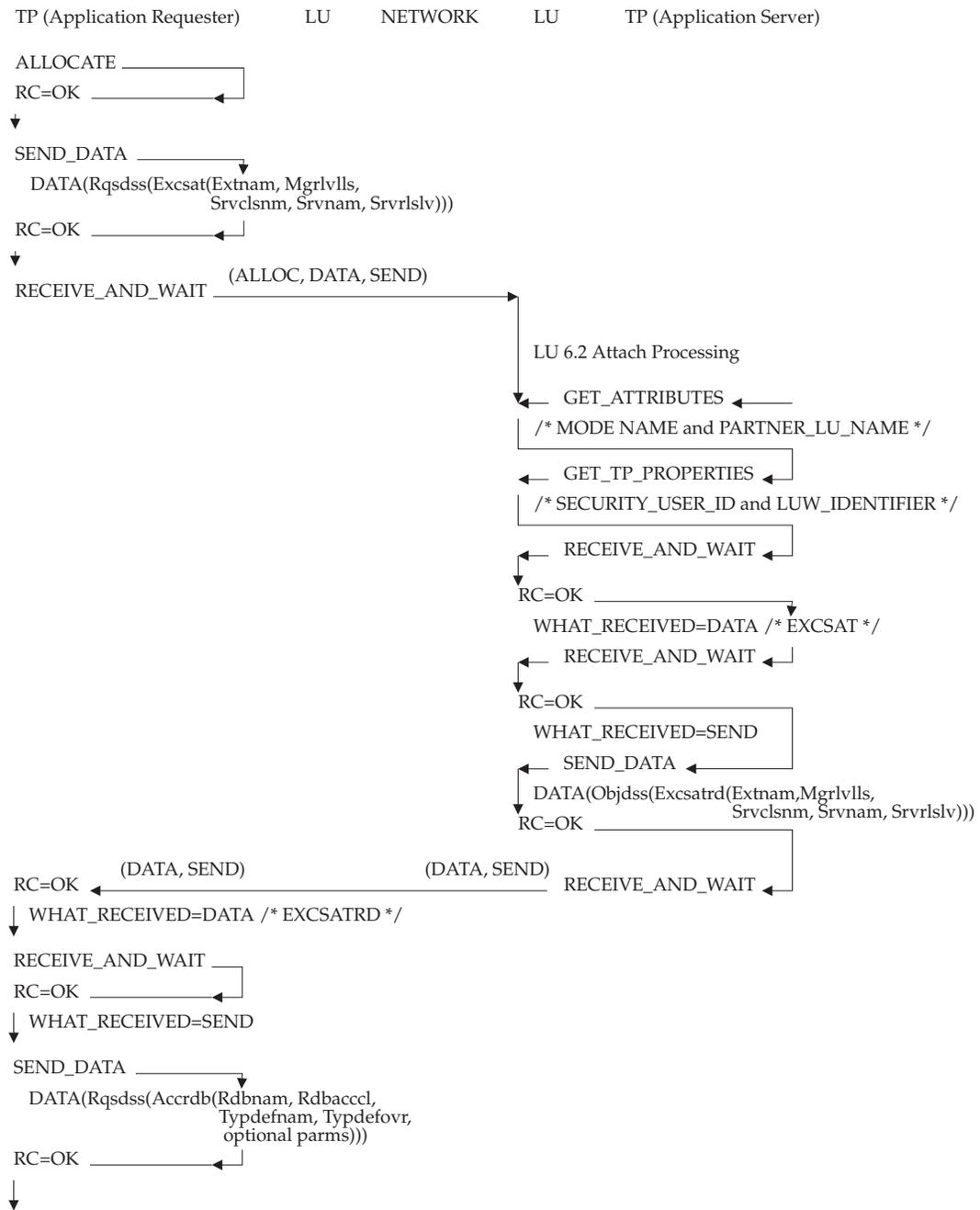
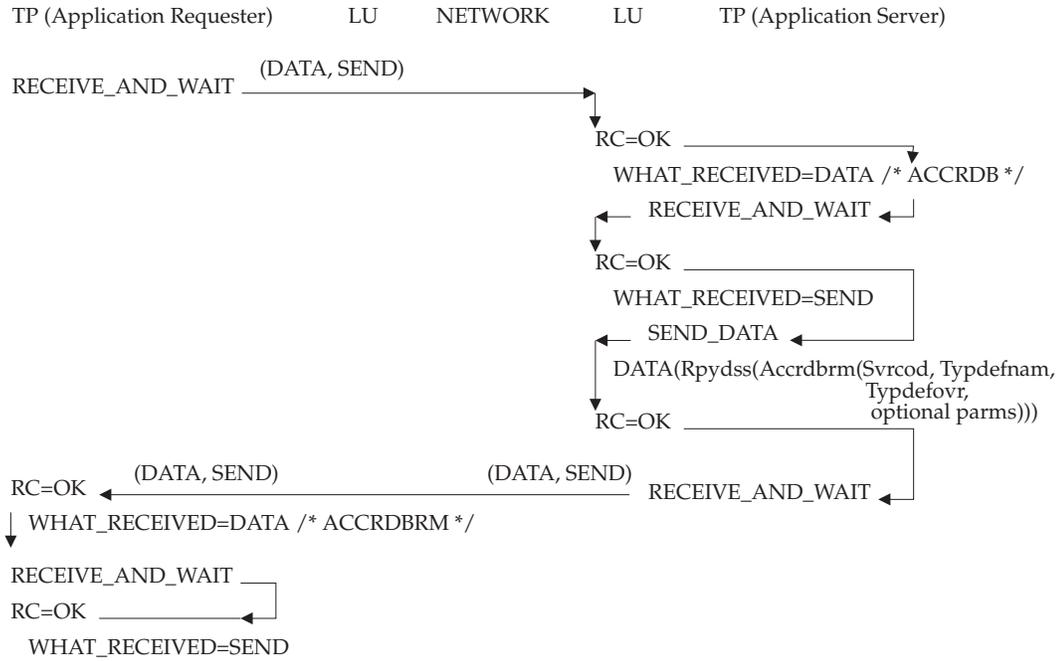


Figure 12-1 DRDA Initialization Flows with LU 6.2 Security (Part 1)



**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 586).



Figure 12-3 DRDA Initialization Flows with DCE Security (Part 1)



Figure 12-4 DRDA Initialization Flows with DCE Security (Part 2)

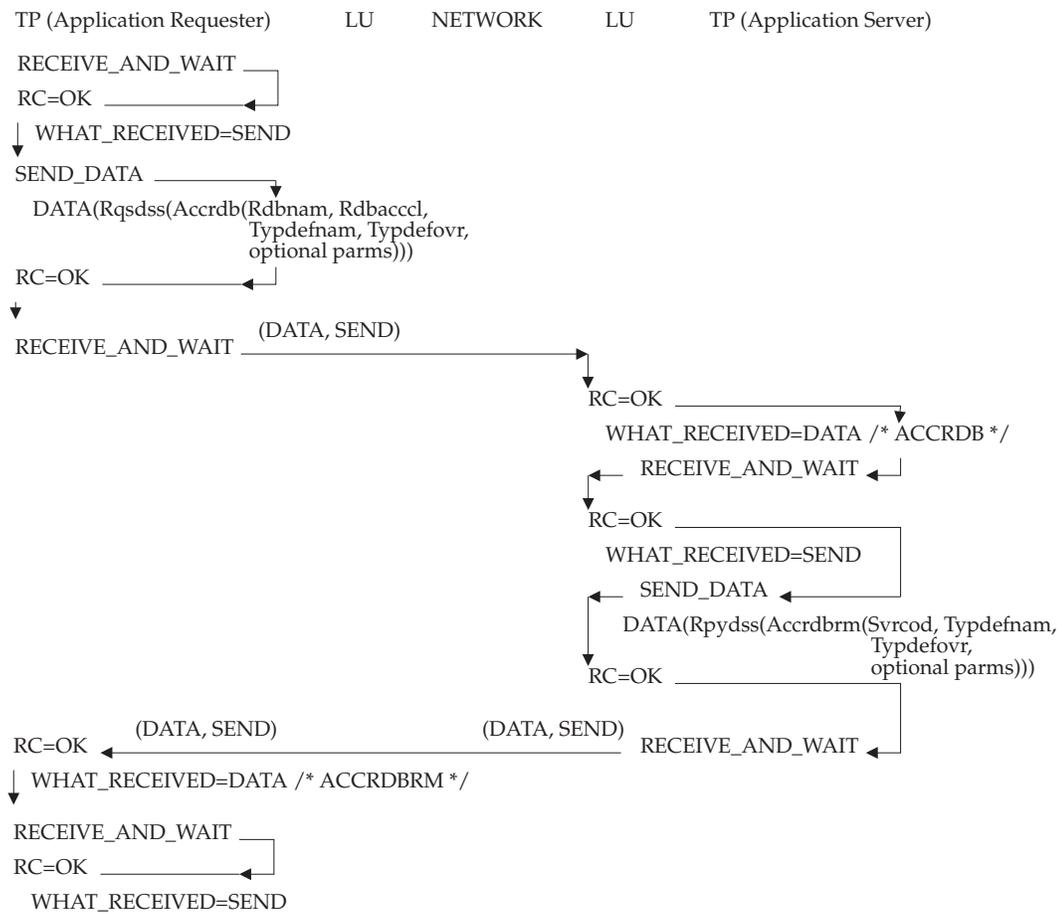


Figure 12-5 DRDA Initialization Flows with DCE Security (Part 3)

## 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. 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](#) 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](#) through [Section 4.4.11](#) 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 depicts DDM command processing using the LU 6.2 synchronous wait protocol verbs. DRDA also permits asynchronous wait protocols. Figure 12-6 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](#) for further detail about DRDA DDM command sequences.

A bind flow is shown in [Figure 12-6](#) (on page 591).

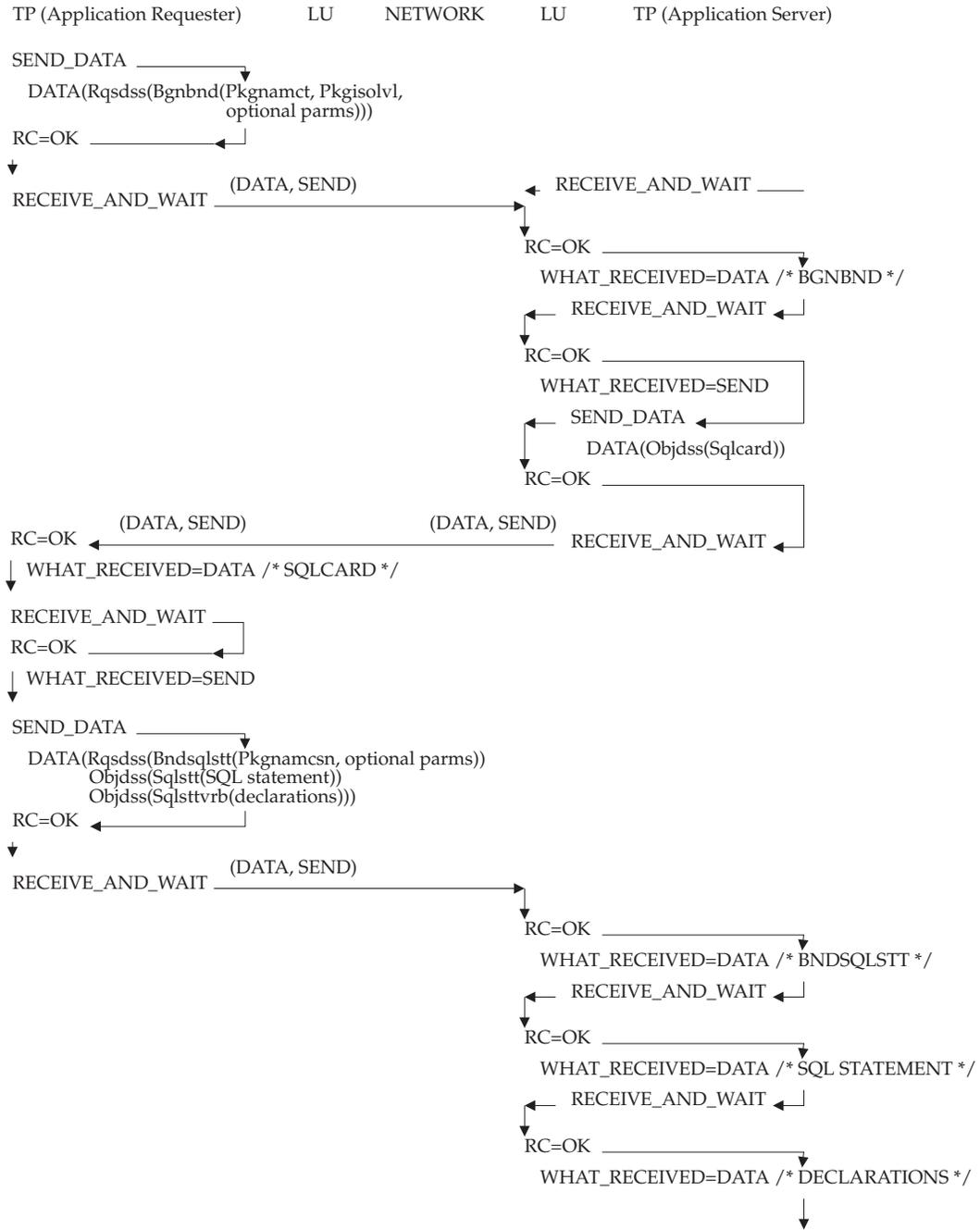


Figure 12-6 DRDA Bind Flows (Part 1)

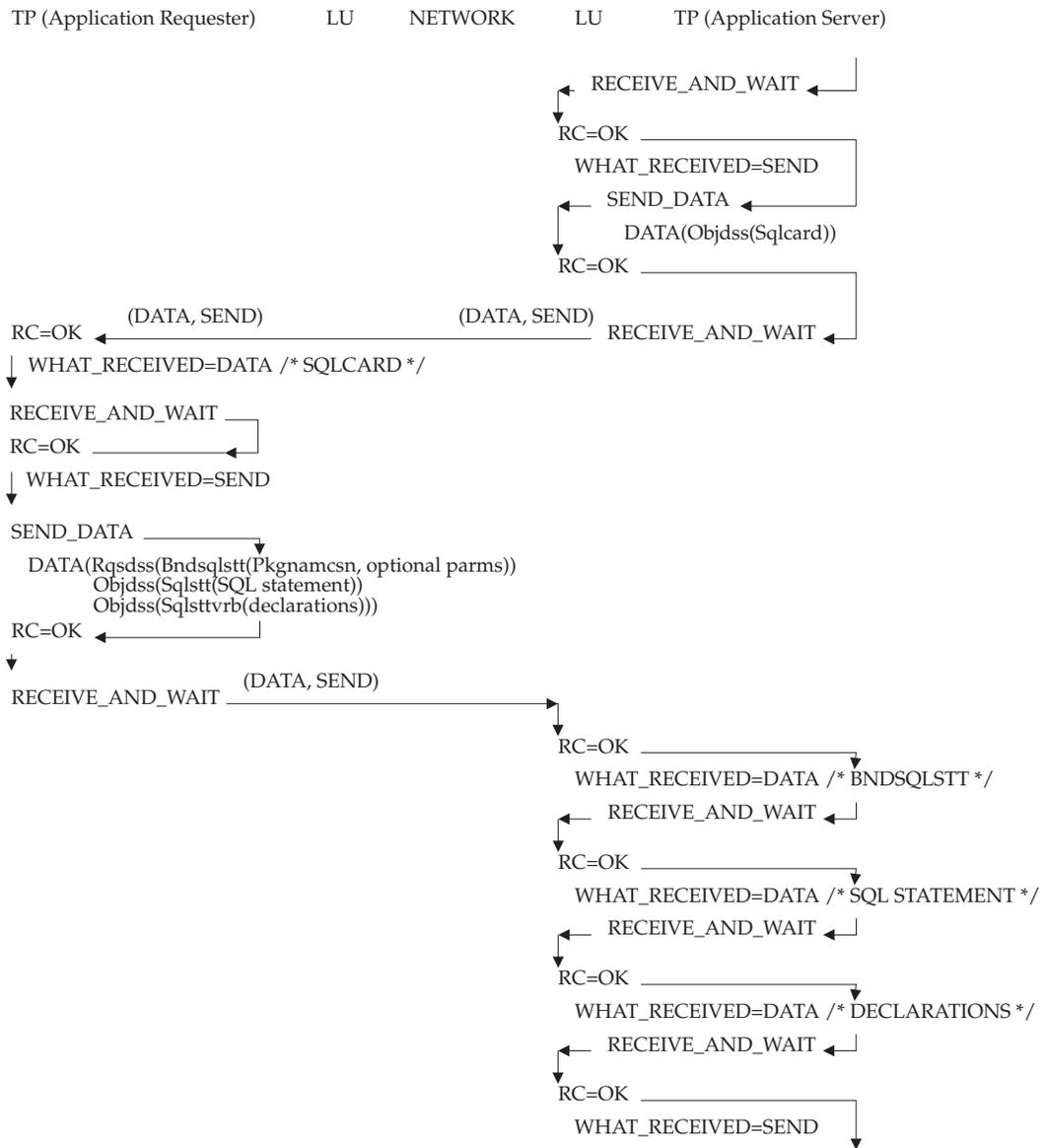
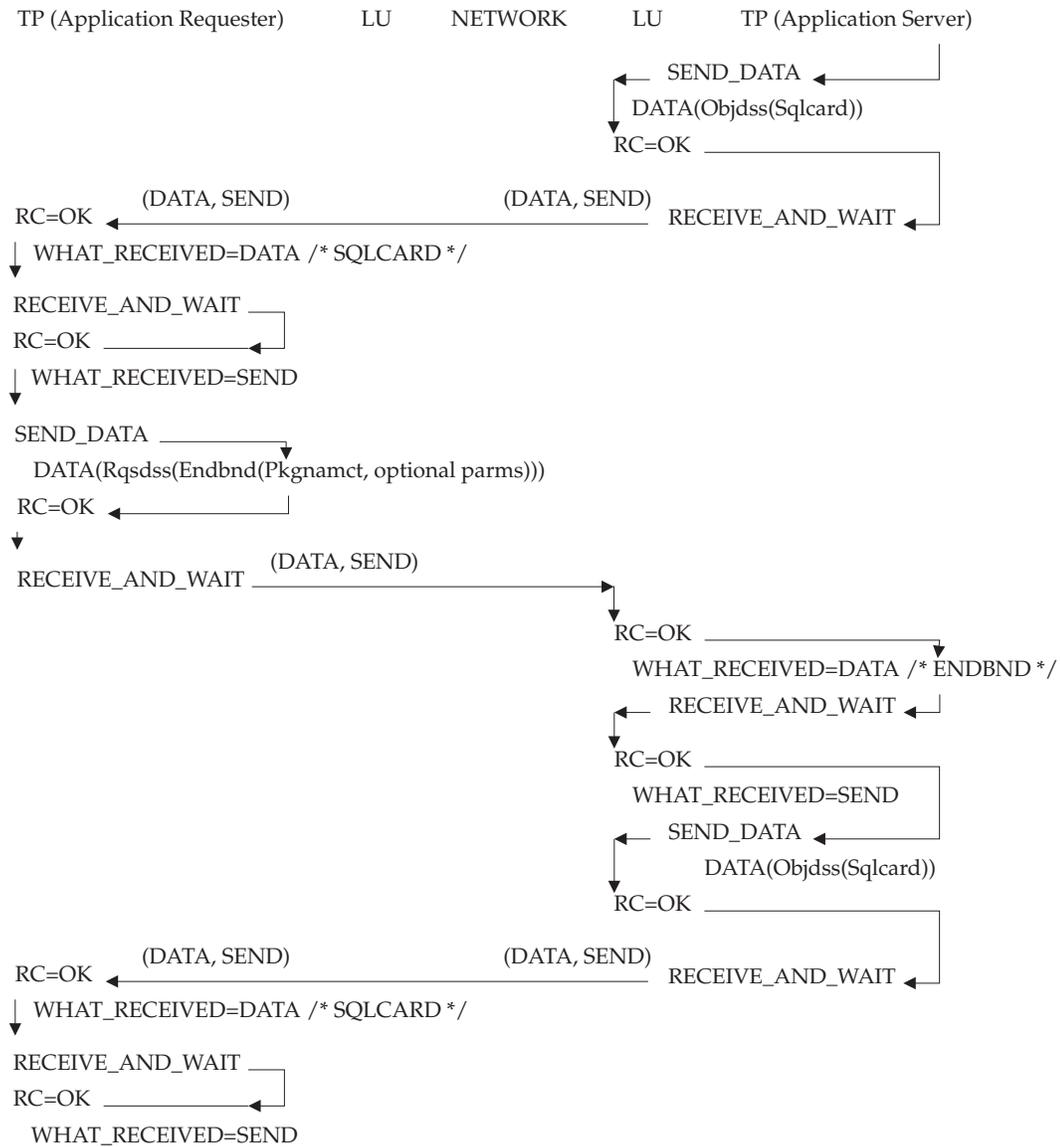


Figure 12-7 DRDA Bind Flows (Part 2)



**Figure 12-8** DRDA Bind Flows (Part 3)

12.7.2.4 SQL Statement Execution Flows

Figure 12-9 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.

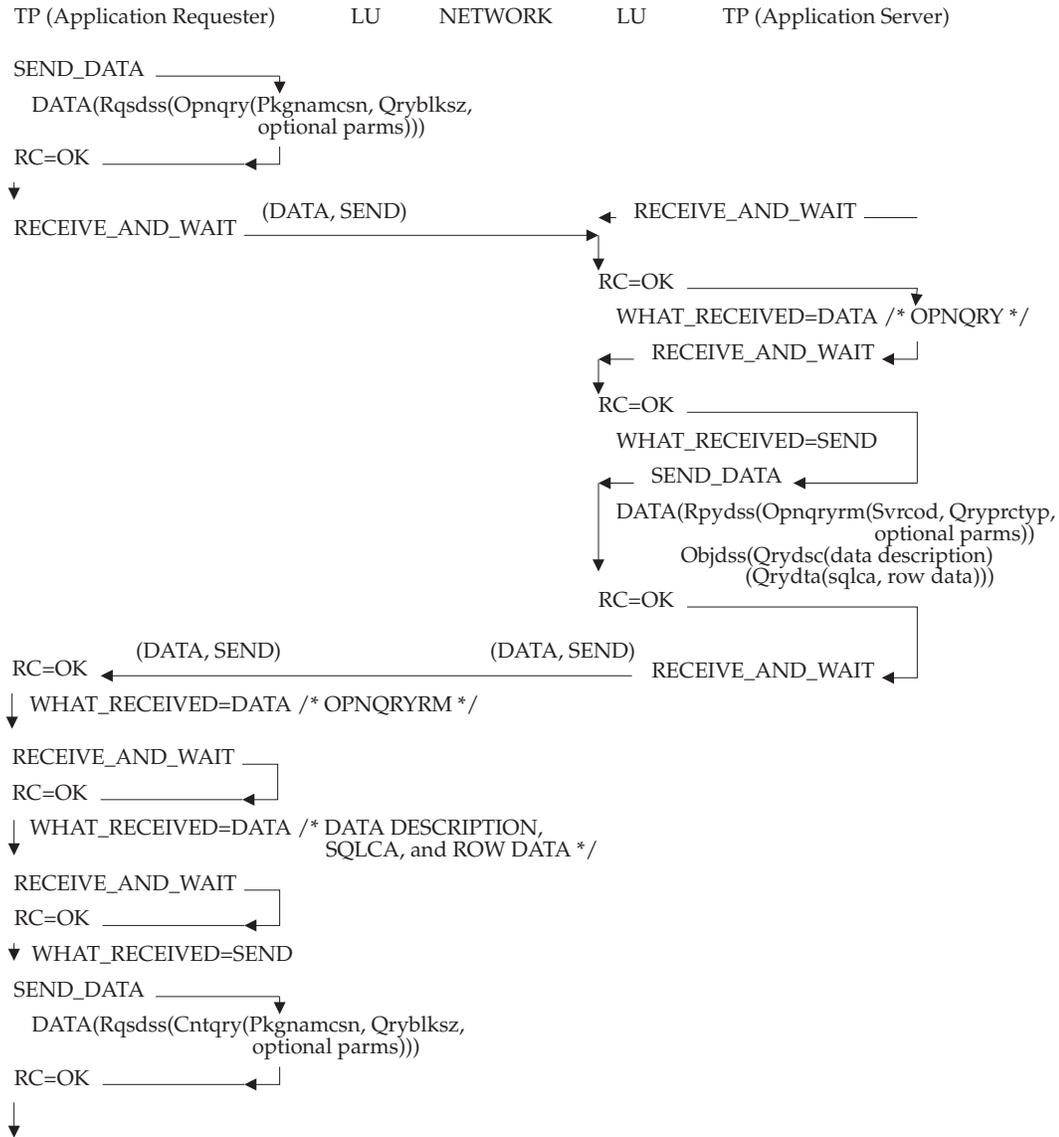
Individual LU 6.2 SEND\_DATA verbs at the application requester transmit each of the DDM request data stream structures for OPNQRY and CNTQRY. Individual LU 6.2 RECEIVE\_AND\_WAIT verbs at the application requester then receive the DDM object data stream structure and reply message responses for the 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](#) for further detail about DRDA DDM command sequences.

[Figure 12-9](#) shows the SQL statement execution flow.



**Figure 12-9** DRDA SQL Statement Execution Flows (Part 1)

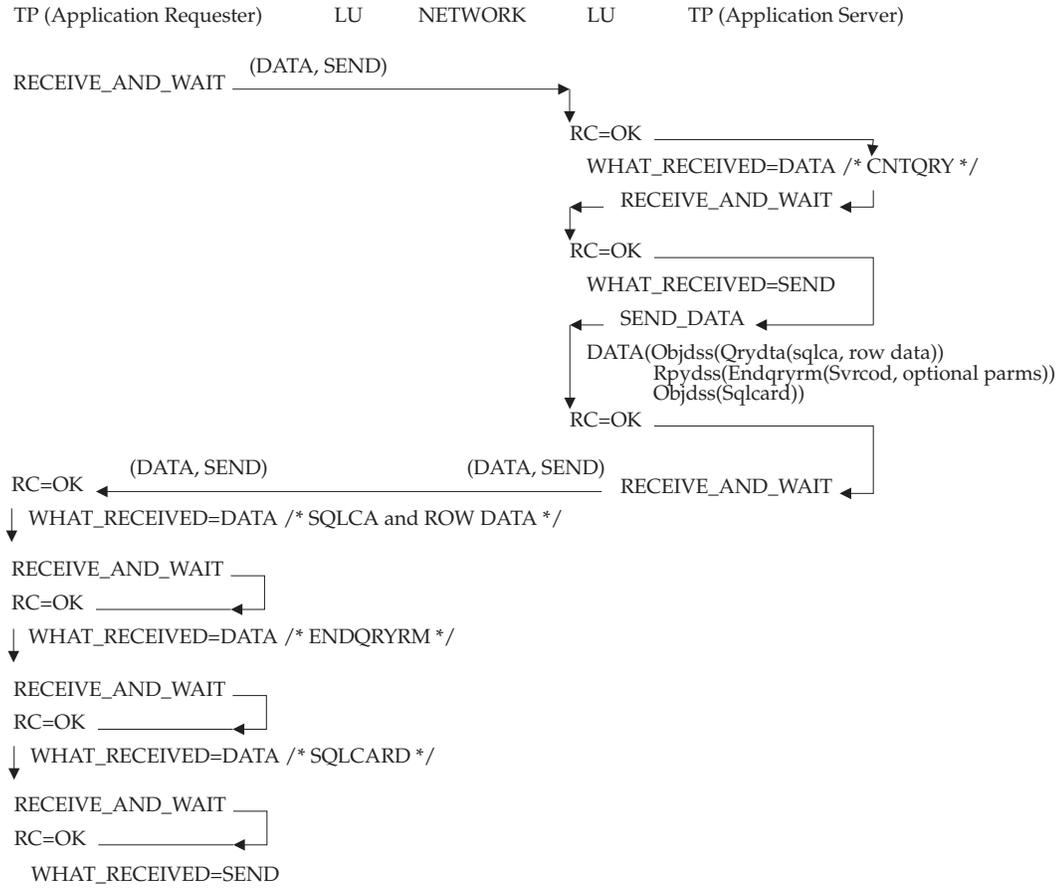


Figure 12-10 DRDA SQL Statement Execution Flows (Part 2)

### 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.

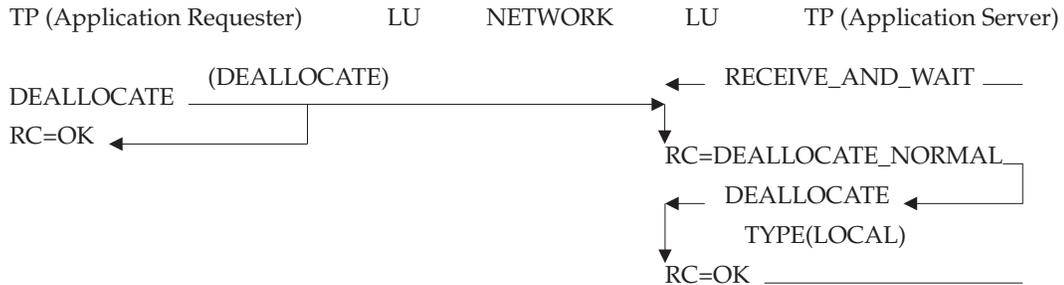
**SYNCPT**

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.

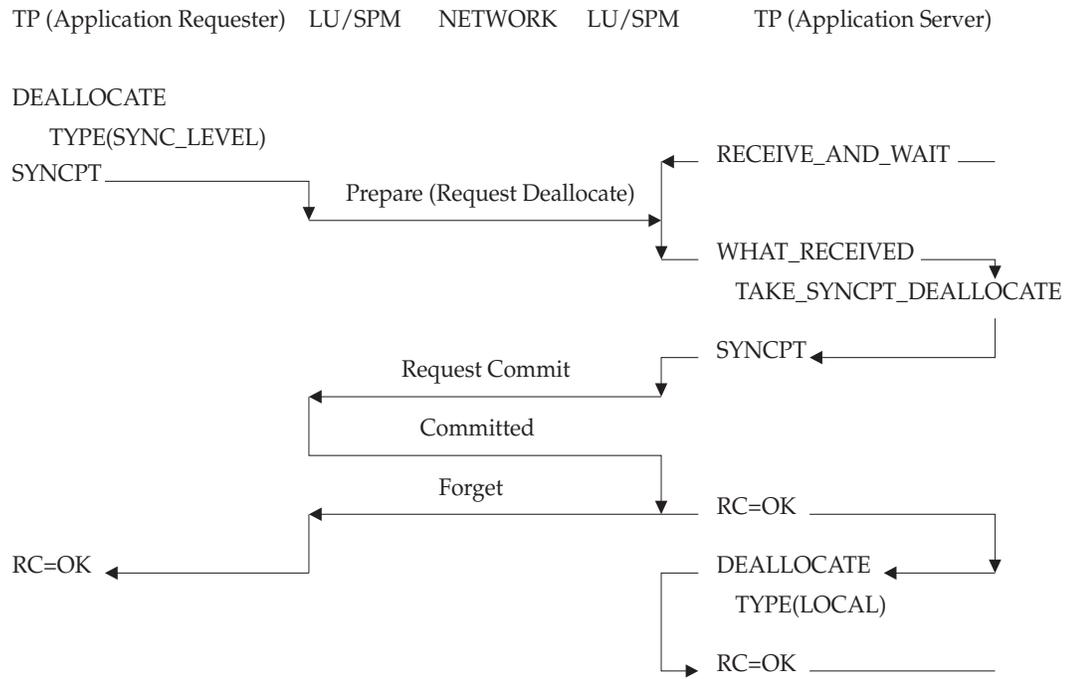
A RECEIVE\_AND\_WAIT at the application server receives the deallocate request, which causes local deallocation of the conversation. [Figure 12-11](#) shows the termination flow on a SYNC\_LEVEL(NONE) conversation.



**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](#) 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**.



**Figure 12-12** Actual Flow: Termination Flows on SYNC\_LEVEL(SYNCPT) Conversation

**12.7.4 Commit Flows on SYNC\_LEVEL(NONE) Conversations**

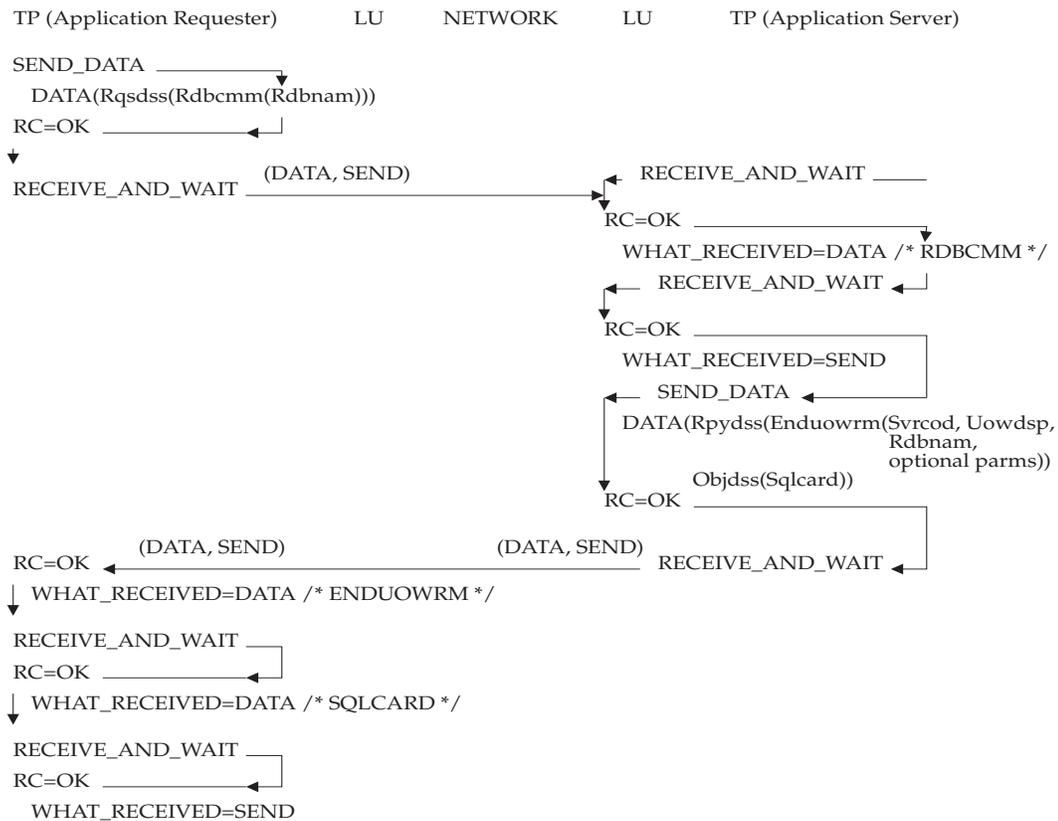
The physical flow of information for commit processing on SYNC\_LEVEL(NONE) conversations consists of a sequence of LU 6.2 verbs containing DDM commands, and DDM reply messages.

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](#) for further detail about DRDA DDM command sequences.

Figure 12-13 shows the commit execution flow.



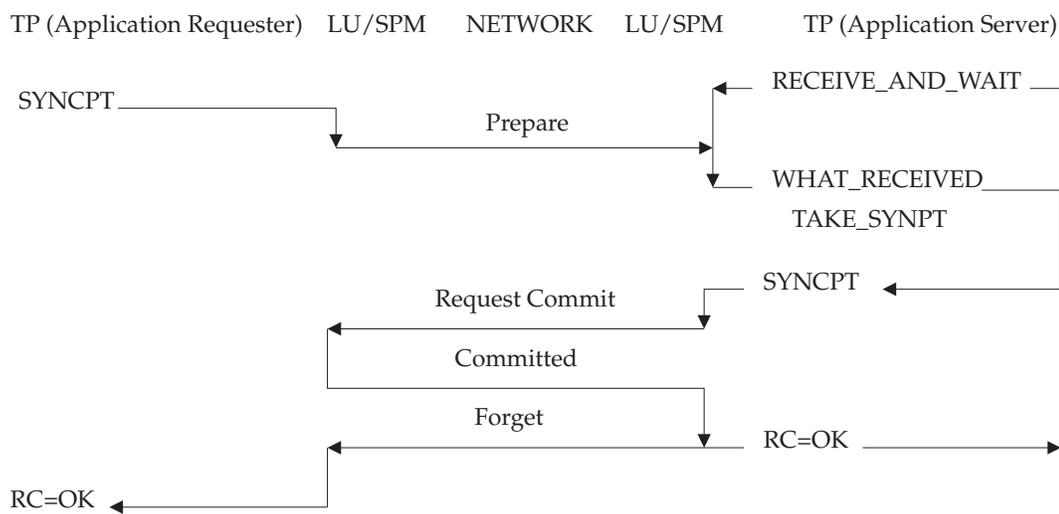
**Figure 12-13** Commit Flow for a SYNC\_LEVEL(NONE) Conversation

**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 and replace RDBCMM with RDBRLLBCK.

**12.7.6 Commit Flows on SYNC\_LEVEL(SYNCPT) Conversations**

Figure 12-14 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**.

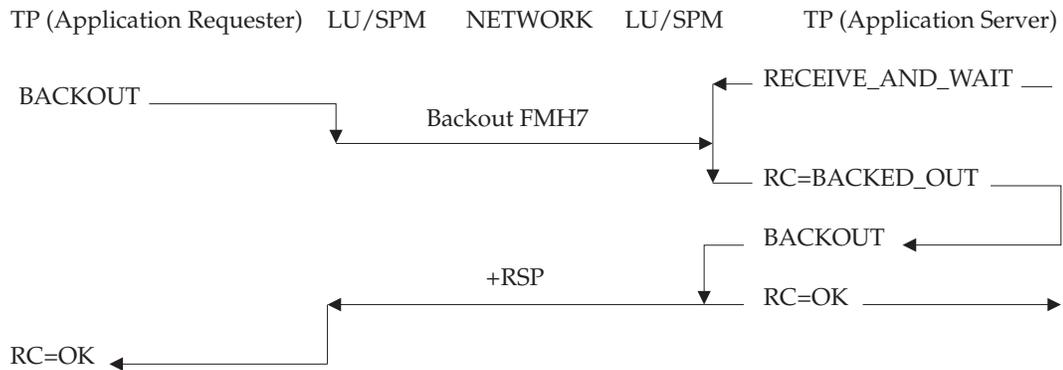


**Figure 12-14** Actual Flow: Commit Flow on a SYNC\_LEVEL(SYNCPT) Conversation

### 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**.

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:

1. Reject any subsequent SQL request from the application.
2. 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 11.3.2.2](#) 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 580)) 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 a reply message that falls into the category of the alerts defined in Table 11-1 (on page 547). 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 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 427).

### 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 or INTRDBRQS 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 7.14](#) (on page 458).

## 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 7.5](#) for setting the *uowdsp* parameter when multiple commit and/or rollbacks occur in a stored procedure. See Rule 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 (see [Section 7.14](#) (on page 458)). 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

<b>SV1 Usage</b>	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.
<b>SV8 Usage</b>	The SNA LUWID or <i>crrtkn</i> or the ACCRDB must be present in the alert, in the supporting data information, and in diagnostic information.
<b>SV9 Usage</b>	Using distributed unit of work protocols, an application requester must send a correlation token to the application server at ACCRDB using the <i>crrtkn</i> 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 <i>crrtkn</i> 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.

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

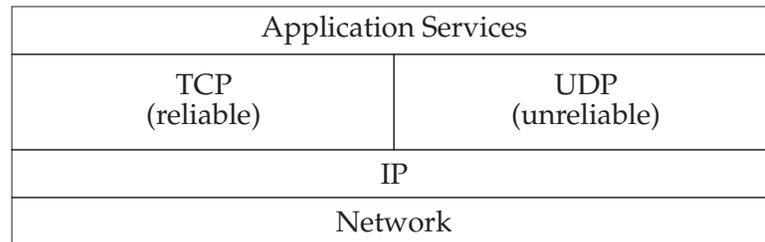
This chapter assumes some familiarity with TCP/IP and the sockets interface. The sockets interface is used only as a convenience to model the functionality level and calls to drive the TCP/IP protocols. With a general exposure to these topics, it should be possible to understand DRDA's use of TCP/IP. 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**.

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** 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](#) (on page 612). 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:

<b>Socket</b>	Creates an end point (socket) for communication.
<b>Close</b>	Closes a socket.
<b>Bind</b>	Establishes a local address for a socket.
<b>Connect</b>	Initiates a TCP/IP connection on a socket.
<b>Listen</b>	Listens for TCP/IP connection requests on a socket.
<b>Accept</b>	Accepts a TCP/IP connection on a socket.
<b>Write</b>	Sends data on a TCP/IP connection.
<b>Read</b>	Receives data on a TCP/IP connection.
<b>Getpeername</b>	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](#) and [Section 6.1.1](#) for 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 TCPCMNI term in the DDM Reference, which discusses initiation of TCP/IP connections.

#### 13.5.1.1 Initialization Flows

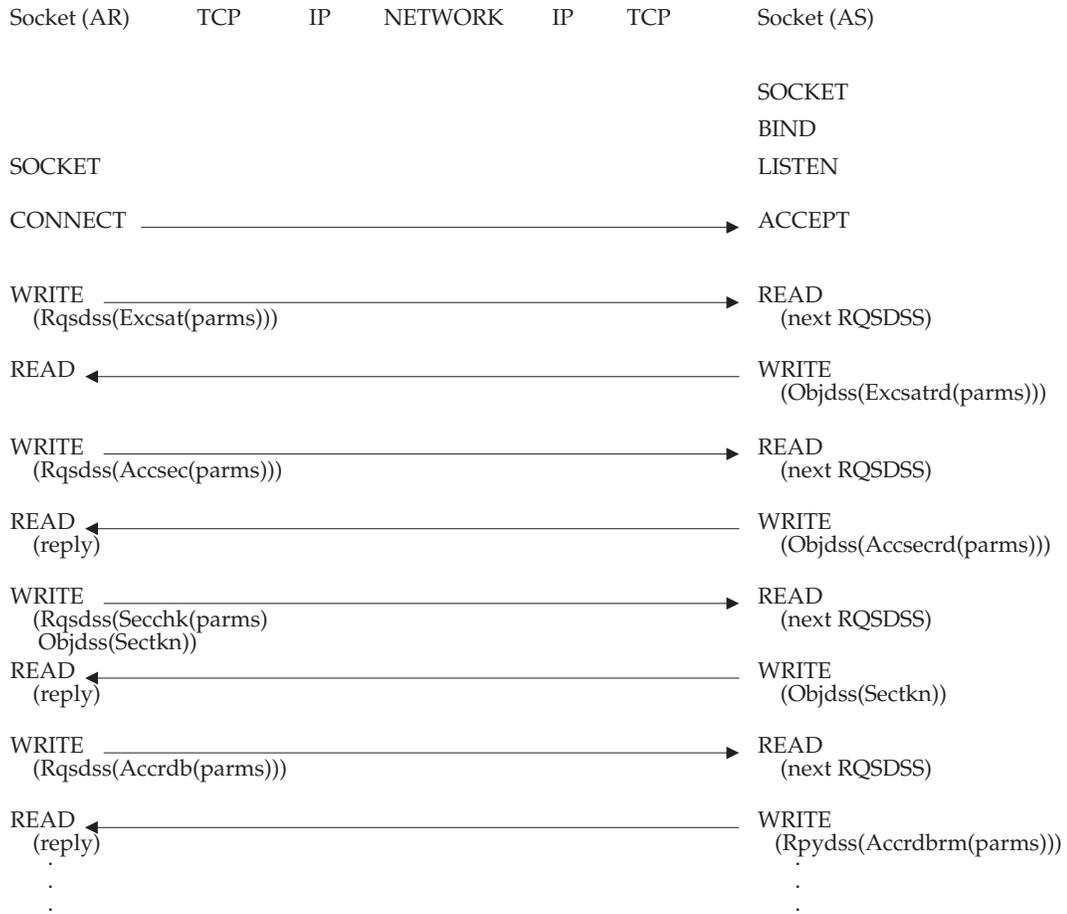
The physical flow of information consists of a sequence of socket calls containing DDM commands. [Figure 13-2](#) depicts the initialization flows while using DRDA-defined user ID and password security or DCE security mechanisms.

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](#) for further detail about DRDA DDM command sequences.

The DRDA TCP/IP initialization flow with negotiation for security mechanisms consists of the following:



**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](#) 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.

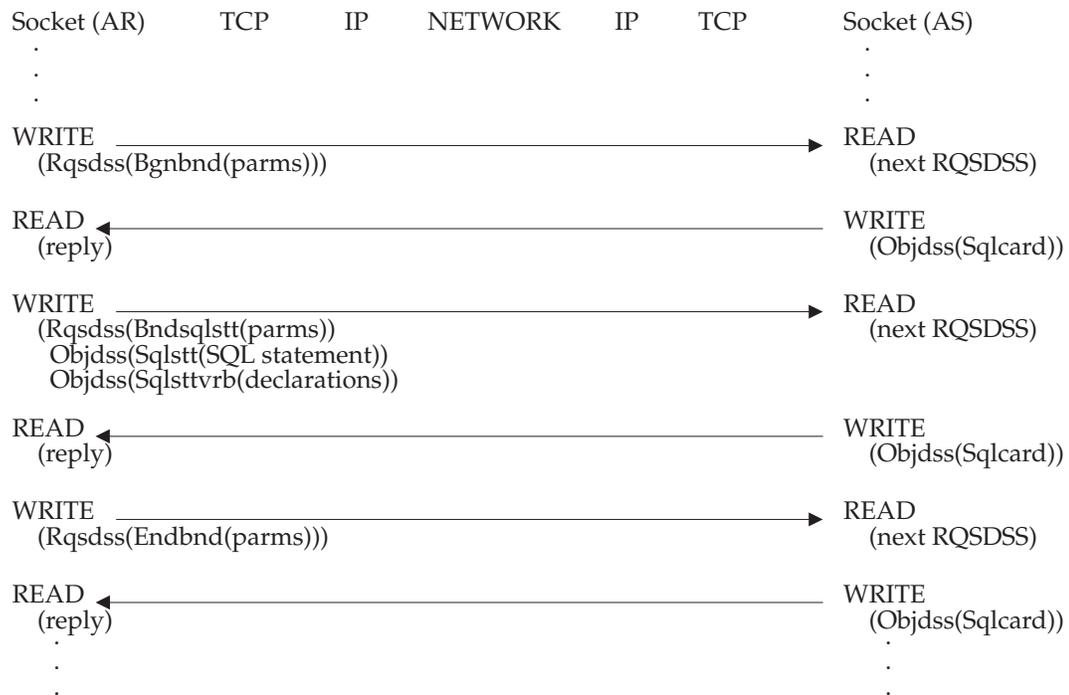
[Figure 13-3](#) depicts DDM command processing using socket interface calls. [Figure 13-3](#) assumes that DDM command chaining is not being used.

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](#) for further detail about DRDA DDM command sequences.

A bind flow is shown in [Figure 13-3](#) (on page 617).



**Figure 13-3** DRDA Bind Flows on TCP/IP

### 13.5.2.2 SQL Statement Execution Flows

**Figure 13-4** depicts DDM command processing using socket interface calls.

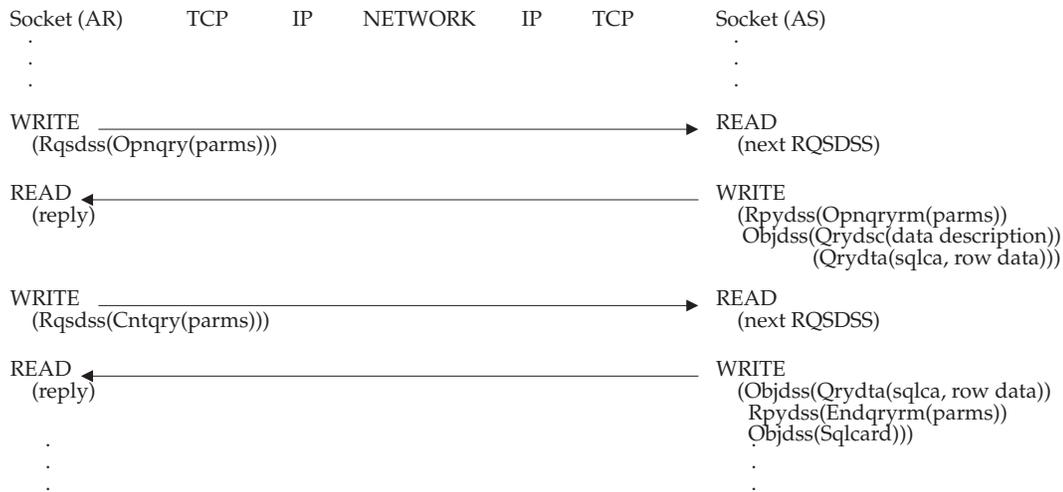
The physical flow of information consists of a sequence of packets containing DDM commands, FD:OCA data descriptors, FD:OCA data, and DDM reply messages.

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** for further detail about DRDA DDM command sequences.

**Figure 13-4** shows the SQL statement execution flow.



**Figure 13-4** DRDA SQL Statement Execution Flows on TCP/IP

### 13.5.3 Terminating a Connection

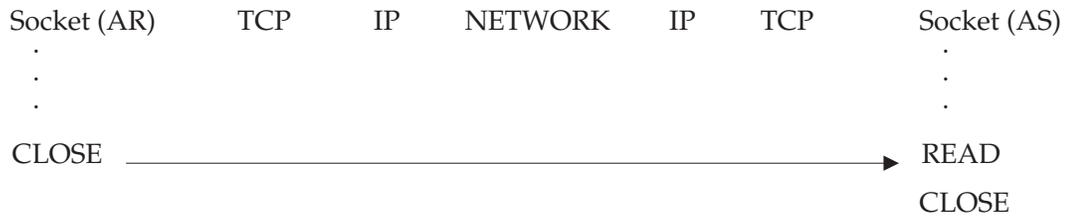
Terminate connection processing closes a socket associated with the TCP/IP connection. Under normal circumstances, only an application requester initiates termination of the socket. In error situations, an application server can also initiate the termination of the socket.

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.

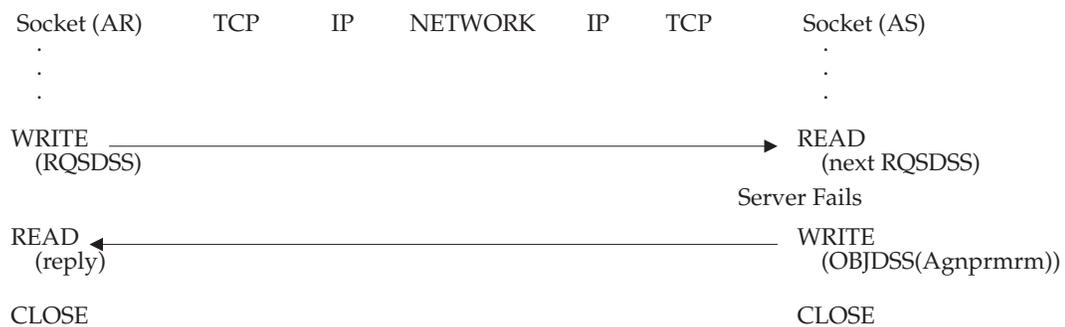
On a TCP/IP connection, the application server receives an indication the socket is terminated. The termination includes an implied rollback at the application server. It is the responsibility of the application server to ensure a rollback during local termination processing at the application server.

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](#) 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.

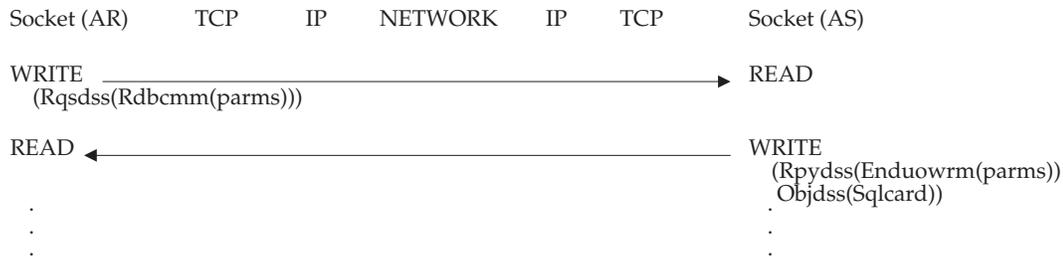
#### 13.5.4.1 Remote Unit of Work

**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](#) for further detail about DRDA DDM command sequences.

Figure 13-7 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 the SYNCPTOV term in the DDM Reference for a definition of the DRDA Level 3 2-phase command sequences and logging requirements. [Figure 13-8](#) 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.

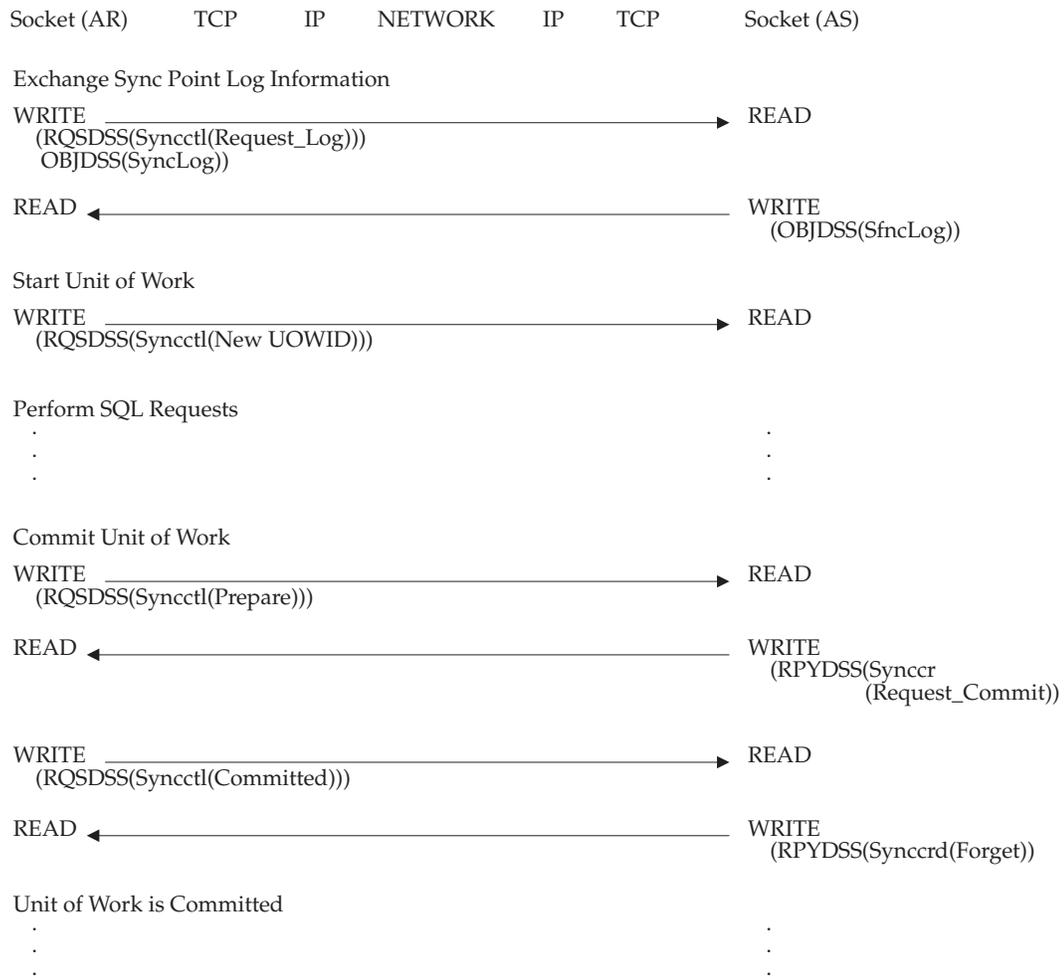
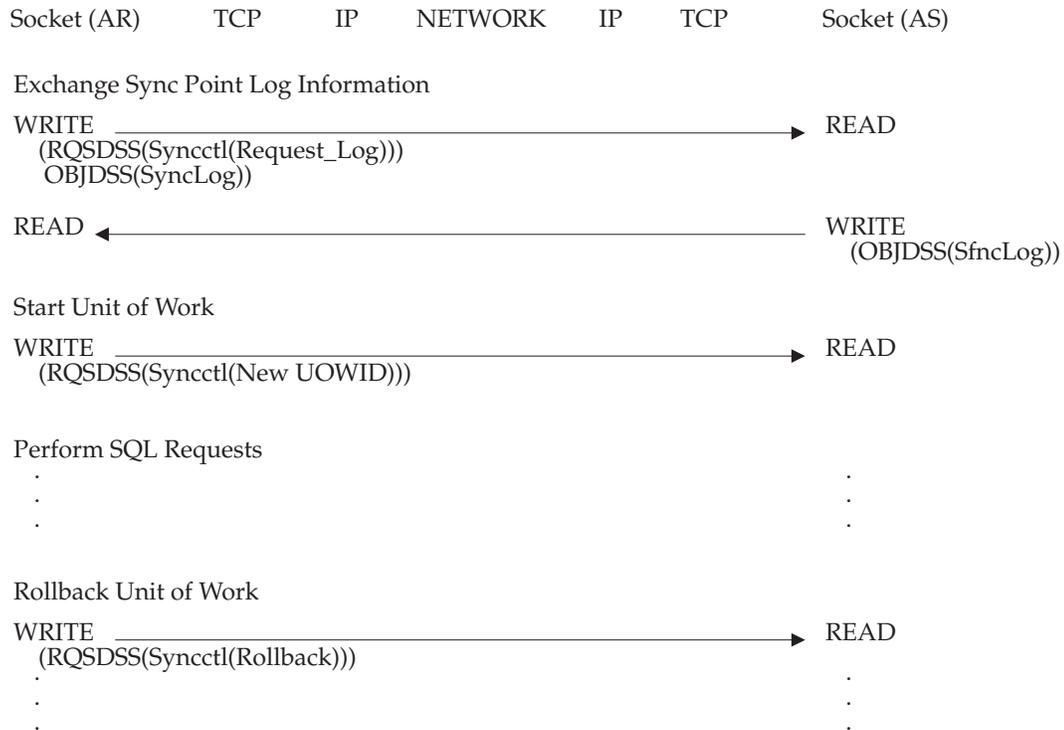


Figure 13-8 TCP/IP Distributed Unit of Work Commit Flow

Rollback Flows: The application requester invokes the DDM Sync Point Manager to coordinate the rollback. Figure 13-9 shows the one message rollback execution flow. The sync point manager rolls back 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 rolls back 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:

1. Reject any subsequent SQL request from the application.
2. 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:

1. For IPv4 network addresses:

```
x.yz
where:
x 8-byte character representation of the 4-byte IP address
  (internal hexadecimal form) of the application requester
. delimiter
y 4-byte character representation of the 2-byte socket address
  (internal hexadecimal form) 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*.
2. For IPv6 network addresses:

```
x.y.z
where:
x a variable 39-byte character representation of the readable
```

```

        (colon separated) 16-byte IP address of the application
        requester
    .   delimiter
    y   a variable 8-byte character representation of the readable
        (decimal value) 2-byte socket address of the application
        requester
    .   delimiter
    z   12-byte character value (possibly a clock value) that makes the
        correlation value unique

```

The specific rules for the display of a correlation value generated are:

1. The application requester must generate a fully displayable CRRTKN, which requires no conversion by the application server to a displayable format.
2. The *x* component must be a readable IP address including the colons. For example:

```

2002:91B:679:15:202:55FF:FE3A:2F03
::FE3A:2F03

```

3. The *y* component must be a readable port value (e.g., 65535).

The format of the LUWID/UOWID is not being extended to accommodate IPv6 addressing due to two-phase commit recovery implications, especially as they relate to “hopping” to down-level systems. However, since the CRRTKN is strictly defined for correlation purposes, as opposed to recovery purposes, the CRRTKN is extended to accommodate IPv6 addresses.

### 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 427).

### 13.6.2.1 TCP/IP Usage of Connection Allocation Rules

<b>CA2 Usage</b>	Connections between an application requester and an application server must have the following socket options: <ul style="list-style-type: none"> <li>• SO_KEEPALIVE: keep connection alive to provide timely detection of a broken connection.</li> <li>• SO_LINGER: linger on close if data present to allow detection of a broken connection.</li> </ul>
<b>CA3 Usage</b>	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 7.5](#) for setting the *uowdsp* parameter when multiple commit and/or rollbacks occur in a stored procedure. See Rule CR6 for the SQLSTATEs to return.

#### 13.6.2.3 TCP/IP Usage of Security (SE Rules)

**SE2 Usage** The application server must support SECMGR Level 5 and above to be able to obtain the verified end-user name associated with the TCP/IP connection. DRDA Level 3 and above, therefore, requires one of the 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.

#### 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.



## 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 A.1 shows the relationship of types of distribution (Remote Unit of Work and Distributed Unit of Work) to the DDM managers. Section A.2 defines the DDM commands, replies, and parameters in relationship to the DDM manager and in relationship to the DRDA level.

### A.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 A-1** DDM Manager Relationship to DRDA Level

Manager	DRDA Remote Unit of Work	DRDA Distributed Unit of Work	DRDA Level 3	DRDA Transactional Processing Interface
AGENT	3	3 or 4	3, 4, or 5	3 or 4
CCSID	0 or ccsid#	0 or ccsid#	0 or ccsid#	0 or ccsid#
CMNAPPC	0 or 3	3	0 or 3	0
CMNSYNCPT	0 or 4	4	0 or 4	0
CMNTCPIP	0, 5, or 8	0, 5 or 8	0, 5, or 8	5 or 8
RDB	3	3	3	3
SQLAM	3	4	3, 4, or 5	4
SYNCPTMGR	0, 4, 5, or 7	4, 5, or 7	0, 4, or 5	0
SECMGR	1	1	5	1
XAMGR	0	0	0	7

## A.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 Optionality (OC Rules) in [Section 7.11](#) (on page 453).

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 A-2** ABNUOWRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rdbnam	Required	Required	Required	Required	Required	Required
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

## The ACCRDB Command

Table A-3 ACCRDB Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Armcrr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Crrtkn	Optional	Required	Required	Required	Required	Required
Diaglvl	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Prddta	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable
Prdid	Required	Required	Required	Required	Required	Required
Rdbacccl	Required	Required	Required	Required	Required	Required
Rdbalwupd	Optional	Optional	Optional	Optional	Optional	Optional
Rdbnam	Required	Required	Required	Required	Required	Required
Sttdeccl	Optional	Optional	Optional	Optional	Optional	Optional
Sttstrdel	Optional	Optional	Optional	Optional	Optional	Optional
Trust	Undefined	Undefined	Undefined	Undefined	Undefined	Optional
Typdefnam	Required	Required	Required	Required	Required	Required
Typdefovr	Required	Required	Required	Required	Required	Required
Unpupdalw	Undefined	Undefined	Undefined	Undefined	Undefined	Optional

Table A-4 Reply Objects for the ACCRDB Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Sqlcard	Optional	Optional	Optional	Optional	Optional	Optional
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

## The ACCRDBRM Reply Message

Table A-5 ACCRDBRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Crrtkn	Required/Conditional	Required/Conditional	Required/Conditional	Required/Conditional	Required/Conditional	Required/Conditional
Pkgdfcst	Optional/Conditional	Optional/Conditional	Optional/Conditional	Optional/Conditional	Optional/Conditional	Optional/Conditional
Prdid	Required	Required	Required	Required	Required	Required
Rdbintipaddr	Undefined	Undefined	Undefined	Undefined	Optional/Ignorable	Optional/Ignorable
Rdbintsnaaddr	Undefined	Undefined	Undefined	Undefined	Optional/Ignorable	Optional/Ignorable
Rdbinttkn	Undefined	Undefined	Undefined	Undefined	Optional/Ignorable	Optional/Ignorable
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Srvlst	Undefined	Undefined	Optional/Ignorable	Optional/Ignorable	Optional/Ignorable	Optional/Ignorable
Svrcod	Required	Required	Required	Required	Required	Required
Typdefnam	Required	Required	Required	Required	Required	Required
Typdefovr	Required	Required	Required	Required	Required	Required
Userid	Optional/Conditional	Optional/Conditional	Optional/Conditional	Optional/Conditional	Optional/Conditional	Optional/Conditional

## The ACCSEC Command

Table A-6 ACCSEC Command Instance Variables

Instance Variable	SECMGR Level 5	SECMGR Level 6	SECMGR Level 7	SECMGR Level 8
Encalg	Undefined	Undefined	Optional	Optional
Enckeylen	Undefined	Undefined	Optional	Optional
Plginid	Undefined	Undefined	Optional/ Conditional	Optional/ Conditional
Plginnm	Undefined	Undefined	Optional/ Conditional	Optional/ Conditional
Secmec	Required	Required	Required	Required
Secmgrnm	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable
Sectkn	Undefined	Optional/ Conditional	Optional/ Conditional	Optional/ Conditional

Table A-7 Reply Objects for the ACCSEC Command

Reply Object	SECMGR Level 5	SECMGR Level 6	SECMGR Level 7	SECMGR Level 8
Accsecrd	Required	Required	Required	Required
Kersecpl	Undefined	Undefined	Optional	Optional
Plginlst	Undefined	Undefined	Optional	Optional

## The ACCSECRD Reply Object

Table A-8 ACCSECRD Reply Object Instance Variables

Instance Variable	SECMGR Level 5	SECMGR Level 6	SECMGR Level 7	SECMGR Level 8
Encalg	Undefined	Undefined	Optional	Optional
Enckeylen	Undefined	Undefined	Optional	Optional
Secchkcd	Undefined	Optional	Optional	Optional
Secmec	Required	Required	Required	Required
Sectkn	Undefined	Optional/ Conditional	Optional/ Conditional	Optional/ Conditional

**The AGNPRMRM Reply Message****Table A-9** AGNPRMRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The BGNATMCHN Command****Table A-10** BGNATMCHN Command Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rtnsetstt	Undefined	Undefined	Undefined	Undefined	Optional	Optional

## The BGNBND Command

Table A-11 BGNBND Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Bndchkexs	Optional	Optional	Optional	Optional	Optional	Optional
Bndcrtctl	Optional	Optional	Optional	Optional	Optional	Optional
Bndexpopt	Optional	Optional	Optional	Optional	Optional	Optional
Decprc	Optional	Optional	Optional	Optional	Optional	Optional
Dftrdbcol	Optional	Optional	Optional	Optional	Optional	Optional
Dgriopr1	Undefined	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable
Pkgathopt	Optional	Optional	Optional	Optional	Optional	Optional
Pkgathrul	Undefined	Undefined	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable
Pkgdfdcc	Optional	Optional	Optional	Optional	Optional	Optional
Pkgdfcst	Optional	Optional	Optional	Optional	Optional	Optional
Pkgrplopt	Optional	Optional	Optional	Optional	Optional	Optional
Pkgisolvl	Required	Required	Required	Required	Required	Required
Pkgnamct	Required	Required	Required	Required	Required	Required
Pkgownid	Optional	Optional	Optional	Optional	Optional	Optional
Pkgrplvrs	Optional	Optional	Optional	Optional	Optional	Optional
Prpsttkp	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryblkctl	Optional	Optional	Optional	Optional	Optional	Optional
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Rdbrlsopt	Optional	Optional	Optional	Optional	Optional	Optional
Sttdatfmt	Optional	Optional	Optional	Optional	Optional	Optional
Sttdecdel	Optional	Optional	Optional	Optional	Optional	Optional
Sttimfmt	Optional	Optional	Optional	Optional	Optional	Optional
Sttstrdel	Optional	Optional	Optional	Optional	Optional	Optional
Title	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable
Vrsnam	Optional	Optional	Optional	Optional	Optional	Optional

Table A-12 Command Objects for the BGNBND Command

Command Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Bndopt	Undefined	Undefined	Optional	Optional	Optional	Optional

**Table A-13** Reply Objects for the BGNBND Command

<b>Reply Object</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Mgrlvlovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Sqlcard	Required	Required	Required	Required	Required	Required
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

**The BGNBNDRM Reply Message****Table A-14** BGNBNDRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Pkgnamct	Required	Required	Required	Required	Required	Required
Rdbnam	Required	Required	Required	Required	Required	Required
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required
Vrsnam	Required	Required	Required	Required	Required	Required

**The BNDCPY Command****Table A-15** BNDCPY Command Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 8</b>
Bndchkexs	Optional
Bndcrtctl	Optional
Bndexpopt	Optional
Decprc	Optional
Dftrdbcol	Optional
Dgrioprl	Optional/Ignorable
Pkgathopt	Optional
Pkgathrul	Optional/Ignorable
Pkgdftcc	Optional
Pkgdftcst	Optional
Pkgisolvl	Optional
Pkgnam	Required
Pkgownid	Optional
Pkgrplopt	Optional
Pkgrplvrs	Optional
Prpsttkp	Optional
Qryblkctl	Optional
Rdbnam	Optional
Rdbrlsopt	Optional
Rdbsrcolid	Required
Sttdatfmt	Optional
Sttdecdel	Optional
Sttimfmt	Optional
Sttstrdel	Optional
Title	Optional/Ignorable
Vrsnam	Optional

**The BNDDPLY Command****Table A-16** BNDDPLY Command Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 8</b>
Dftrdbcol	Optional
Pkgnam	Required
Pkgownid	Optional
Pkgrplopt	Optional
Pkgrplvrs	Optional
Rdbnam	Optional
Rdbsrccolid	Required
Vrsnam	Required

**The BNDSQLSTT Command****Table A-17** BNDSQLSTT Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Bndsttasm	Optional	Optional	Optional	Optional	Optional	Optional
Pkgnamcsn	Required	Required	Required	Required	Optional	Optional
Pkgsn	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Sqlsttnbr	Optional	Optional	Optional	Optional	Optional	Optional

**Table A-18** Command Objects for the BNDSQLSTT Command

Command Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Sectknovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Sqlstt	Required	Required	Required	Required	Required	Required
Sqlsttvrb	Optional	Optional	Optional	Optional	Optional	Optional
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

**Table A-19** Reply Objects for the BNDSQLSTT Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Mgrlvlovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Sqlcard	Required	Required	Required	Required	Required	Required
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

## The CLSQRY Command

Table A-20 CLSQRY Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Cmdsrcid	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Monitor	Optional	Optional	Optional	Optional	Optional	Optional
Pkgnamcsn	Required	Required	Required	Required	Optional	Optional
Pkgsn	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryclsrsls	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryinsid	Undefined	Undefined	Undefined	Undefined	Required	Required
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional

Table A-21 Reply Objects for the CLSQRY Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Mgrlvlovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Monitor	Optional	Optional	Optional	Optional	Optional	Optional
Sqlcard	Required	Required	Required	Required	Required	Required
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

**The CMDATHRM Reply Message****Table A-22** CMDATHRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional

**The CMDCHKRM Reply Message****Table A-23** CMDCHKRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The CMDNSPRM Reply Message****Table A-24** CMDNSPRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Codpnt	Required	Required	Required	Required	Required	Required
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The CMDVTLRM Reply Message****Table A-25** CMDVLTRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rdbnam	Undefined	Required	Required	Required	Required	Required
Srvdgn	Undefined	Optional	Optional	Optional	Optional	Optional
Svrcod	Undefined	Required	Required	Required	Required	Required

**The CMMRQSRM Reply Message****Table A-26** CMMRQSRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Cmmtyp	Undefined	Required	Required	Required	Required	Required
Rdbnam	Undefined	Required	Required	Required	Required	Required
Srvdgn	Undefined	Optional	Optional	Optional	Optional	Optional
Svrcod	Undefined	Required	Required	Required	Required	Required

## The CNTQRY Command

Table A-27 CNTQRY Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Cmdscrid	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Freprvref	Undefined	Undefined	Undefined	Undefined	Undefined	Optional
Maxblkext	Undefined	Undefined	Optional	Optional	Optional	Optional
Monitor	Optional	Optional	Optional	Optional	Optional	Optional
Nbrrow	Undefined	Optional	Optional	Optional	Removed	Undefined
Pkgnamcsn	Required	Required	Required	Required	Optional	Optional
Pkgsn	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryblkrst	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryblksz	Required	Required	Required	Required	Required	Required
Qryinsid	Undefined	Undefined	Undefined	Undefined	Required	Required
Qryrelscr	Undefined	Optional	Optional	Optional	Removed	Undefined
Qryrfrtbl	Undefined	Optional	Optional	Optional	Removed	Undefined
Qryrownbr	Undefined	Optional	Optional	Optional	Optional	Optional
Qryrowset	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryrowsns	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryrtndta	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryscorn	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional

Table A-28 Reply Objects for the CNTQRY Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Mgrlvlovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Monitor	Optional	Optional	Optional	Optional	Optional	Optional
Qrydta	Optional	Optional	Optional	Optional	Optional	Optional
Sectknovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional
Sqlcard	Required/ Conditional	Required/ Conditional	Required/ Conditional	Required/ Conditional	Required/ Conditional	Required/ Conditional

**The DRPPKG Command****Table A-29** DRPPKG Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Pkgid	Undefined	Undefined	Undefined	Undefined	Undefined	Optional
Pkgidany	Undefined	Undefined	Undefined	Undefined	Undefined	Optional
Pkgnam	Required	Required	Required	Required	Required	Optional
Rdbcolid	Undefined	Undefined	Undefined	Undefined	Undefined	Optional
Rdbcolidany	Undefined	Undefined	Undefined	Undefined	Undefined	Optional
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Vrsnam	Optional	Optional	Optional	Optional	Optional	Optional
Vrsnamany	Undefined	Undefined	Undefined	Undefined	Undefined	Optional

**Table A-30** Reply Objects for the DRPPKG Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Mgrlvlovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Sqlcard	Required	Required	Required	Required	Required	Required
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

## The DSCINVRM Reply Message

Table A-31 DSCINVRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Dscerrcd	Required	Required	Required	Required	Required	Required
Fdodsc	Required	Required	Required	Required	Required	Required
Fdodscoff	Required	Required	Required	Required	Required	Required
Fdoprwoff	Required	Required	Required	Required	Required	Required
Fdotrwoff	Required	Required	Required	Required	Required	Required
Rdbnam	Required	Required	Required	Required	Required	Required
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The DSCRDBTBL Command****Table A-32** DSCRDBTBL Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Monitor	Optional	Optional	Optional	Optional	Optional	Optional
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional

**Table A-33** Command Objects for the DSCRDBTBL Command

Command Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Sqlobjnam	Required	Required	Required	Required	Required	Required
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

**Table A-34** Reply Objects for the DSCRDBTBL Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Mgrlvlovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Monitor	Optional	Optional	Optional	Optional	Optional	Optional
Sectknovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Sqldata	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional
Sqldard	Required/ Mutually- exclusive	Required/ Mutually- exclusive	Required/ Mutually- exclusive	Required/ Mutually- exclusive	Required/ Mutually- exclusive	Required/ Mutually- exclusive
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

## The DSCSQLSTT Command

Table A-35 DSCSQLSTT Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Cmdsrcid	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Monitor	Optional	Optional	Optional	Optional	Optional	Optional
Pkgnamcsn	Required	Required	Required	Required	Optional	Optional
Pkgsn	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryinsid	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Typsqlda	Undefined	Undefined	Undefined	Optional	Optional	Optional

Table A-36 Reply Objects for the DSCSQLSTT Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Mgrlvlovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Monitor	Optional	Optional	Optional	Optional	Optional	Optional
Sectknovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Sqlcard	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional
Sqldard	Required/ Mutually- exclusive	Required/ Mutually- exclusive	Required/ Mutually- exclusive	Required/ Mutually- exclusive	Required/ Mutually- exclusive	Required/ Mutually- exclusive
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

**The DTAMCHRM Reply Message****Table A-37** DTAMCHRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rdbnam	Required	Required	Required	Required	Required	Required
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The ENDATMCHN Command****Table A-38** ENDATMCHN Command Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Endchntyp	Undefined	Undefined	Undefined	Undefined	Optional	Optional

**The ENDBND Command****Table A-39** ENDBND Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Maxsctnbr	Optional	Optional	Optional	Optional	Optional	Optional
Pkgnamct	Required	Required	Required	Required	Required	Required
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional

**Table A-40** Reply Objects for the ENDBND Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Mgrlvlovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Sqlcard	Required	Required	Required	Required	Required	Required
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

**The ENDQYRM Reply Message****Table A-41** ENDQYRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The ENDUOWRM Reply Message****Table A-42** ENDUOWRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Release	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required
Uowdsp	Required	Required	Required	Required	Required	Required

**The EXCSAT Command****Table A-43** EXCSAT Command Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Extnam	Required	Required	Required	Required	Required	Required
Mgrlvls	Required	Required	Required	Required	Required	Required
Spvnam	Optional	Optional	Optional	Optional	Optional	Optional
Srvclsnm	Required	Required	Required	Required	Required	Required
Srvnam	Required	Required	Required	Required	Required	Required
Srvrlslv	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable

**The EXCSATRD Reply Object****Table A-44** EXCSATRD Reply Object Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Extnam	Required	Required	Required	Required	Required	Required
Mgrlvls	Required	Required	Required	Required	Required	Required
Srvclsnm	Required	Required	Required	Required	Required	Required
Srvnam	Required	Required	Required	Required	Required	Required
Srvrslv	Optional	Optional	Optional	Optional	Optional	Optional

## The EXCSQLIMM Command

Table A-45 EXCSQLIMM Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Cmdsrcid	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Monitor	Optional	Optional	Optional	Optional	Optional	Optional
Pkgnamcsn	Required	Required	Required	Required	Optional	Optional
Pkgsn	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryinsid	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Rdbcmtok	Undefined	Optional	Optional	Optional	Optional	Optional
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Rtnsetstt	Undefined	Undefined	Undefined	Undefined	Optional	Optional

Table A-46 Command Objects for the EXCSQLIMM Command

Command Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Sectknovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Sqlstt	Required	Required	Required	Required	Required	Required
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

Table A-47 Reply Objects for the EXCSQLIMM Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Mgrlvlovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Monitor	Optional	Optional	Optional	Optional	Optional	Optional
Sqlcard	Required	Required	Required	Required	Required	Required
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

**The EXCSQLSET Command****Table A-48** EXCSQLSET Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Rtnsetstt	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Pkgnamct	Required	Required	Required	Required	Optional	Optional
Pkgsn	Undefined	Undefined	Undefined	Undefined	Optional	Optional

**Table A-49** Command Objects for the EXCSQLSET Command

Command Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Sectknovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Sqlstt	Required	Required	Required	Required	Required	Required
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

**Table A-50** Reply Objects for the EXCSQLSET Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Mgrlvlovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Sqlcard	Required	Required	Required	Required	Required	Required
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

## The EXCSQLSTT Command

Table A-51 EXCSQLSTT Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Atmind	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Cmdsrcid	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Dyndtafmt	Undefined	Undefined	Undefined	Undefined	Undefined	Optional
Maxblkext	Undefined	Undefined	Optional	Optional	Optional	Optional
Maxrslcnt	Undefined	Undefined	Optional	Optional	Optional	Optional
Meddtasz	Undefined	Undefined	Undefined	Undefined	Undefined	Optional/ Ignorable
Monitor	Optional	Optional	Optional	Optional	Optional	Optional
Nbrrow	Undefined	Optional	Optional	Optional	Optional	Optional
Outexp	Optional	Optional	Optional	Optional	Optional	Optional
Pkgnamcsn	Required	Required	Required	Required	Optional	Optional
Pkgsn	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Prcnam	Undefined	Optional	Optional	Optional	Optional	Optional
Qryblksz	Undefined	Undefined	Optional	Optional	Optional	Optional
Qryclsimp	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryclsrsls	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryextdtasz	Undefined	Undefined	Undefined	Undefined	Undefined	Optional/ Ignorable
Qryinsid	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryrowset	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Rdbcmtok	Undefined	Optional	Optional	Optional	Optional	Optional
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Rslsetflg	Undefined	Undefined	Optional	Optional	Optional	Optional
Rtnsetstt	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Rtnsqlda	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Smldtasz	Undefined	Undefined	Undefined	Undefined	Undefined	Optional/ Ignorable
Typsqlda	Undefined	Undefined	Undefined	Undefined	Optional	Optional

Table A-52 Command Objects for the EXCSQLSTT Command

Command Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Sectknovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Sqllda	Optional	Optional	Optional	Optional	Optional	Optional
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

**Table A-53** Reply Objects for the EXCSQLSTT Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Mgrlvlovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Monitor	Optional	Optional	Optional	Optional	Optional	Optional
Qrydta	Undefined	Undefined	Optional	Optional	Optional	Optional
Qrydsc	Undefined	Undefined	Optional	Optional	Optional	Optional
Sectknovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Sqlcard	Required/ Mutually- exclusive	Required/ Mutually- exclusive	Required/ Mutually- exclusive	Required/ Mutually- exclusive	Required/ Mutually- exclusive	Required/ Mutually- exclusive
Sqlcinrd	Undefined	Undefined	Optional	Optional	Optional	Optional
Sqltdard	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional
Sqlrslrd	Undefined	Undefined	Optional	Optional	Optional	Optional
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

## The GETNXTCHK Command

Table A-54 GETNXTCHK Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Cmdsrcid	Undefined	Undefined	Undefined	Undefined	Undefined	Optional
Frerefopt	Undefined	Undefined	Undefined	Undefined	Undefined	Optional
Getnxtlen	Undefined	Undefined	Undefined	Undefined	Undefined	Optional
Getnxtref	Undefined	Undefined	Undefined	Undefined	Undefined	Required
Pkgnamcsn	Undefined	Undefined	Undefined	Undefined	Undefined	Required
Qryinsid	Undefined	Undefined	Undefined	Undefined	Undefined	Required
Refrst	Undefined	Undefined	Undefined	Undefined	Undefined	Optional

Table A-55 Reply Objects for the GETNXTCHK Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Mgrlvlovr	Undefined	Undefined	Undefined	Undefined	Undefined	Optional
Monitor	Undefined	Undefined	Undefined	Undefined	Undefined	Optional
Exttda	Undefined	Undefined	Undefined	Undefined	Undefined	Required
Typdefnam	Undefined	Undefined	Undefined	Undefined	Undefined	Optional
Typdefovr	Undefined	Undefined	Undefined	Undefined	Undefined	Optional
Sqlcard	Undefined	Undefined	Undefined	Undefined	Undefined	Required/ Conditional

**The INTRDBRQS Reply Message****Table A-56** INTRDBRQS Command Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rdbinttkn	Undefined	Undefined	Undefined	Undefined	Required	Required
Rdbnam	Optional	Optional	Required	Required	Required	Required
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The INTTKNRM Reply Message****Table A-57** INTTKNRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rdbnam	Undefined	Undefined	Undefined	Undefined	Required	Required
Rdbinttkn	Undefined	Undefined	Undefined	Undefined	Required	Required
Srvdgn	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Svrcod	Undefined	Undefined	Undefined	Undefined	Required	Required

**The MGRDEPRM Reply Message****Table A-58** MGRDEPRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Deperrcd	Required	Required	Required	Required	Required	Required
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The MGRLVLRM Reply Message****Table A-59** MGRLVLRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Mgrlvla	Required	Required	Required	Required	Required	Required
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The OBJNSPRM Reply Message****Table A-60** OBJNSPRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Codpnt	Required	Required	Required	Required	Required	Required
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The OPNQFLRM Reply Message****Table A-61** OPNQFLRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rdbnam	Required	Required	Required	Required	Required	Required
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

## The OPNQRY Command

Table A-62 OPNQRY Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Cmdsrcid	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Dupqryok	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Dyndtafmt	Undefined	Undefined	Undefined	Undefined	Undefined	Optional
Maxblkext	Undefined	Undefined	Optional	Optional	Optional	Optional
Meddtasz	Undefined	Undefined	Undefined	Undefined	Undefined	Optional/ Ignorable
Monitor	Optional	Optional	Optional	Optional	Optional	Optional
Pkgnamcsn	Required	Required	Required	Required	Optional	Optional
Pkgsn	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryblkctl	Optional	Optional	Optional	Optional	Optional	Optional
Qryblksz	Required	Required	Required	Required	Required	Required
Qryclsimp	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryclsrsl	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryextdtasz	Undefined	Undefined	Undefined	Undefined	Undefined	Optional/ Ignorable
Qryrowset	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Rtnsqlda	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Smldtasz	Undefined	Undefined	Undefined	Undefined	Undefined	Optional/ Ignorable
Typsqlda	Undefined	Undefined	Undefined	Undefined	Optional	Optional

Table A-63 Command Objects for the OPNQRY Command

Command Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Sectknovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Sqllda	Optional	Optional	Optional	Optional	Optional	Optional
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

**Table A-64** Reply Objects for the OPNQRY Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Exttda	Undefined	Undefined	Undefined	Undefined	Undefined	Optional/Conditional
Mgrlvlovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Monitor	Optional	Optional	Optional	Optional	Optional	Optional
Qrydsc	Required	Required	Required	Required	Required	Required
Qrydta	Optional/Conditional	Optional/Conditional	Optional/Conditional	Optional/Conditional	Optional/Conditional	Optional/Conditional
Sectknovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Sqlcard	Optional	Optional	Optional	Optional	Optional	Optional
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

## The OPNQRYRM Reply Message

Table A-65 OPNQRYRM Reply Message Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Dyndtafmt	Undefined	Undefined	Undefined	Undefined	Undefined	Optional
Qryattscr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryattset	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryattsns	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryattupd	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryblkfct	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryblktyp	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Qryinsid	Undefined	Undefined	Undefined	Undefined	Required	Required
Qryprctyp	Required	Required	Required	Required	Required	Required
Sqlcsrhd	Optional	Optional	Optional	Optional	Optional	Optional
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The PKGBNARM Reply Message****Table A-66** PKGBNARM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rdbnam	Required	Required	Required	Required	Required	Required
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The PKGBPARAM Reply Message****Table A-67** PKGBPARAM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rdbnam	Required	Required	Required	Required	Required	Required
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The PRCCNVRM Reply Message****Table A-68** PRCCNVRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Prcnvc	Required	Required	Required	Required	Required	Required
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The PRMNSPRM Reply Message****Table A-69** PRMNSPRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Codpnt	Required	Required	Required	Required	Required	Required
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

## The PRPSQLSTT Command

Table A-70 PRPSQLSTT Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Bufinsind	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Cmdsrcid	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Monitor	Optional	Optional	Optional	Optional	Optional	Optional
Pkgnamcsn	Required	Required	Required	Required	Optional	Optional
Pkgsn	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Rtnsqlda	Optional	Optional	Optional	Optional	Optional	Optional
Typsqlda	Undefined	Undefined	Undefined	Undefined	Optional	Optional

Table A-71 Command Objects for the PRPSQLSTT Command

Command Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Sectknovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Sqlattr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Sqlstt	Required	Required	Required	Required	Required	Required
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

Table A-72 Reply Objects for the PRPSQLSTT Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Mgrlvlovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Monitor	Optional	Optional	Optional	Optional	Optional	Optional
Sectknovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Sqlcard	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional
Sqldard	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional	Required/ Mutually- exclusive/ Conditional
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

**The QRYNOPRM Reply Message****Table A-73** QRYNOPRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Pkgnamcsn	Required	Required	Required	Required	Required	Required
Rdbnam	Required	Required	Required	Required	Required	Required
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The QRYPOPRM Reply Message****Table A-74** QRYPOPRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Pkgnamcsn	Required	Required	Required	Required	Required	Required
Rdbnam	Required	Required	Required	Required	Required	Required
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The RDBACCRM Reply Message****Table A-75** RDBACCRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rdbnam	Required	Required	Required	Required	Required	Required
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The RDBAFLRM Reply Message****Table A-76** RDBAFLRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rdbnam	Required	Required	Required	Required	Required	Required
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The RDBATHRM Reply Message****Table A-77** RDBATHRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rdbnam	Required	Required	Required	Required	Required	Required
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The RDBCMM Command****Table A-78** RDBCMM Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Release	Undefined	Undefined	Undefined	Undefined	Optional	Optional

**Table A-79** Reply Objects for the RDBCMM Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Mgrlvlovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Sqlcard	Required	Required	Required	Required	Required	Required
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

**The RDBNACRM Reply Message****Table A-80** RDBNACRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rdbnam	Required	Required	Required	Required	Required	Required
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The RDBNFNRM Reply Message****Table A-81** RDBNFNRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rdbnam	Required	Required	Required	Required	Required	Required
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The RDBRLLBCK Command****Table A-82** RDBRLLBCK Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Release	Undefined	Undefined	Undefined	Undefined	Optional	Optional

**Table A-83** Reply Objects for the RDBRLLBCK Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Mgrlvlovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional
Sqlcard	Required	Required	Required	Required	Required	Required
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

**The RDBUPDRM Reply Message****Table A-84** RDBUPDRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rdbnam	Undefined	Required	Required	Required	Required	Required
Srvdgn	Undefined	Optional	Optional	Optional	Optional	Optional
Svrcod	Undefined	Required	Required	Required	Required	Required
Unpupd	Undefined	Undefined	Undefined	Undefined	Undefined	Optional

**The REBIND Command****Table A-85** REBIND Command Instance Variables

Instance Variable	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Bndchkexs	Optional	Optional	Optional	Optional	Optional	Optional
Bndexpopt	Optional	Optional	Optional	Optional	Optional	Optional
Dftrdbcol	Optional	Optional	Optional	Optional	Optional	Optional
Dgriopr1	Undefined	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable
Pkgathrul	Undefined	Undefined	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable	Optional/ Ignorable
Pkgisolvl	Optional	Optional	Optional	Optional	Optional	Optional
Pkgnam	Required	Required	Required	Required	Required	Required
Pkgownid	Optional	Optional	Optional	Optional	Optional	Optional
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Rdbrlsopt	Optional	Optional	Optional	Optional	Optional	Optional
Vrsnam	Optional	Optional	Optional	Optional	Optional	Optional

**Table A-86** Command Objects for the REBIND Command

Command Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Bndopt	Undefined	Undefined	Optional	Optional	Optional	Optional
Mgrlvlovr	Undefined	Undefined	Undefined	Undefined	Optional	Optional

**Table A-87** Reply Objects for the REBIND Command

Reply Object	SQLAM Level 3	SQLAM Level 4	SQLAM Level 5	SQLAM Level 6	SQLAM Level 7	SQLAM Level 8
Sqlcard	Required	Required	Required	Required	Required	Required
Typdefnam	Optional	Optional	Optional	Optional	Optional	Optional
Typdefovr	Optional	Optional	Optional	Optional	Optional	Optional

**The RSCLMTRM Reply Message****Table A-88** RSCLMTRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Prdid	Optional	Optional	Optional	Optional	Optional	Optional
Rdbnam	Required/ Conditional	Required/ Conditional	Required/ Conditional	Required/ Conditional	Required/ Conditional	Required/ Conditional
Rscnam	Optional	Optional	Optional	Optional	Optional	Optional
Rsctyp	Optional	Optional	Optional	Optional	Optional	Optional
Rscod	Optional	Optional	Optional	Optional	Optional	Optional
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The RSLSETRM Reply Message****Table A-89** RSLSETRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Pkgsnlst	Undefined	Undefined	Optional	Optional	Optional	Optional
Srvdgn	Undefined	Undefined	Optional	Optional	Optional	Optional
Svrcod	Undefined	Undefined	Optional	Optional	Optional	Optional

## The SECCHK Command

Table A-90 SECCHK Command Instance Variables

Instance Variable	SECMGR Level 5	SECMGR Level 6	SECMGR Level 7	SECMGR Level 8
Newpassword	Optional/Conditional	Optional/Conditional	Optional/Conditional	Optional/Conditional
Password	Optional/Conditional	Optional/Conditional	Optional/Conditional	Optional/Conditional
Plginid	Undefined	Undefined	Optional/Conditional	Optional/Conditional
Plginnm	Undefined	Undefined	Optional/Conditional	Optional/Conditional
Secmec	Required	Required	Required	Required
Secmgrnm	Optional/Ignorable	Optional/Ignorable	Optional/Ignorable	Optional/Ignorable
Usrid	Optional/Conditional	Optional/Conditional	Optional/Conditional	Optional/Conditional
Usrregnm	Undefined	Undefined	Undefined	Optional
Usrorgid	Undefined	Undefined	Undefined	Optional
Usrsectok	Undefined	Undefined	Undefined	Optional

Table A-91 Command Objects for the SECCHK Command

Command Object	SECMGR Level 5	SECMGR Level 6	SECMGR Level 7	SECMGR Level 8
Sectkn	Optional/Conditional	Optional/Conditional	Optional/Conditional	Optional/Conditional

Table A-92 Reply Objects for the SECCHK Command

Reply Object	SECMGR Level 5	SECMGR Level 6	SECMGR Level 7	SECMGR Level 8
Sectkn	Optional/Conditional	Optional/Conditional	Optional/Conditional	Optional/Conditional

**The SECCHKRM Reply Message****Table A-93** SECCHKRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SECMGR Level 5</b>	<b>SECMGR Level 6</b>	<b>SECMGR Level 7</b>	<b>SECMGR Level 8</b>
Secchkcd	Required	Required	Required	Required
Srvdgn	Optional	Optional	Optional	Optional
Svcerno	Optional/ Ignorable/ Conditional	Optional/ Ignorable/ Conditional	Optional/ Ignorable/ Conditional	Optional/ Ignorable/ Conditional
Svrcod	Required	Required	Required	Required

**The SNDPKT Command****Table A-94** SNDPKT Command Instance Variables

<b>Instance Variable</b>	<b>AGENT Level 3</b>	<b>AGENT Level 4</b>	<b>AGENT Level 5</b>	<b>AGENT Level 6</b>	<b>AGENT Level 7</b>	<b>AGENT Level 8</b>
Respktsz	Undefined	Undefined	Undefined	Undefined	Optional	Optional

**Table A-95** Command Objects for the SNDPKT Command

<b>Command Object</b>	<b>AGENT Level 3</b>	<b>AGENT Level 4</b>	<b>AGENT Level 5</b>	<b>AGENT Level 6</b>	<b>AGENT Level 7</b>	<b>AGENT Level 8</b>
Pkgobj	Undefined	Undefined	Undefined	Undefined	Optional	Optional

**Table A-96** Reply Objects for the SNDPKT Command

<b>Reply Object</b>	<b>AGENT Level 3</b>	<b>AGENT Level 4</b>	<b>AGENT Level 5</b>	<b>AGENT Level 6</b>	<b>AGENT Level 7</b>	<b>AGENT Level 8</b>
Pktobj	Undefined	Undefined	Undefined	Undefined	Required	Required

**The SQLERRRM Reply Message****Table A-97** SQLERRRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The SYNCCTL Command****Table A-98** SYNCCTL Command Instance Variables

<b>Instance Variable</b>	<b>SYNCPTMGR Level 5</b>	<b>SYNCPTMGR Level 7</b>	<b>XAMGR Level 7</b>	<b>XAMGR Level 8</b>
Forget	Optional	Optional	Invalid	Invalid
Rlsconv	Optional	Optional	Optional	Optional
Synctype	Required	Required	Required	Required
Timeout	Undefined	Invalid	Optional	Optional
Uowid	Optional	Optional	Invalid	Invalid
Xaflags	Undefined	Invalid	Required	Required
Xid	Undefined	Optional	Required	Required
Xidshr	Undefined	Optional	Invalid	Invalid

**Table A-99** Command Objects for SYNCCTL

<b>Command Object</b>	<b>SYNCPTMGR Level 5 or 7</b>
Synclog	Optional

**The SYNCCRD Reply Object****Table A-100** SYNCCRD Reply Object Instance Variables

<b>Instance Variable</b>	<b>SYNCPTMGR Level 5</b>	<b>SYNCPTMGR Level 7</b>	<b>XAMGR Level 7</b>	<b>XAMGR Level 8</b>
Prphrclst	Undefined	Invalid	Optional	Optional
Rlsconv	Undefined	Optional	Optional	Optional
Synctype	Required	Required	Invalid	Invalid
Xaretval	Undefined	Invalid	Required	Required

## The SYNCLOG Reply Object

Table A-101 SYNCLOG Reply Object Instance Variables

Instance Variable	SYNCPTMGR Level 5 or 7 and CMNTCPIP
Cnmtkn	Required
Ipaddr	Optional — mutually-exclusive with Snaaddr
IPaddr v6	Undefined
Logname	Required
Logstmp	Required
Rdbnam	Required
Snaaddr	Optional — mutually-exclusive with Ipaddr
Tcphost	Optional — mutually-exclusive with Snaaddr

**The SYNCRSY Command****Table A-102** SYNCRSY Command Instance Variables

<b>Instance Variable</b>	<b>SECMGR Level 5</b>	<b>SECMGR Level 6</b>	<b>SECMGR Level 7</b>	<b>SECMGR Level 8</b>
Rsynctyp	Required	Required	Required	Required
Uowid	Optional	Optional	Optional	Optional
Uowstate	Optional	Optional	Optional	Optional

**Table A-103** Command Objects for SYNCRSY

<b>Command Object</b>	<b>SECMGR Level 5</b>	<b>SECMGR Level 6</b>	<b>SECMGR Level 7</b>	<b>SECMGR Level 8</b>
Synclog	Optional	Optional	Optional	Optional

**The SYNCRRD Reply Object****Table A-104** SYNCRRD Reply Object Instance Variables

<b>Instance Variable</b>	<b>SECMGR Level 5</b>	<b>SECMGR Level 6</b>	<b>SECMGR Level 7</b>	<b>SECMGR Level 8</b>
Rsynctyp	Required	Required	Required	Required
Uowid	Optional	Optional	Optional	Optional
Uowstate	Optional	Optional	Optional	Optional

**Table A-105** Reply Objects for SYNCRRD

<b>Reply Object</b>	<b>SECMGR Level 5</b>	<b>SECMGR Level 6</b>	<b>SECMGR Level 7</b>	<b>SECMGR Level 8</b>
Synclog	Optional	Optional	Optional	Optional

**The SYNTAXRM Reply Message****Table A-106** SYNTAXRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Codpnt	Optional	Optional	Optional	Optional	Optional	Optional
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required
Synerrcd	Required	Required	Required	Required	Required	Required

**The TRGNSPRM Reply Message****Table A-107** TRGNSPRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required

**The VALNSPRM Reply Message****Table A-108** VALNSPRM Reply Message Instance Variables

<b>Instance Variable</b>	<b>SQLAM Level 3</b>	<b>SQLAM Level 4</b>	<b>SQLAM Level 5</b>	<b>SQLAM Level 6</b>	<b>SQLAM Level 7</b>	<b>SQLAM Level 8</b>
Codpnt	Required	Required	Required	Required	Required	Required
Rdbnam	Optional	Optional	Optional	Optional	Optional	Optional
Srvdgn	Optional	Optional	Optional	Optional	Optional	Optional
Svrcod	Required	Required	Required	Required	Required	Required



# Scrollable Cursor Overview

This appendix provides an overview of scrollable cursors as supported in DRDA. It explains the basic concepts, and the key elements and their behavior, required to understand how scrollable cursors operate in DRDA. It concludes with two examples.

## B.1 Key Definitions

### result table

The set of rows specified for the SELECT statement of a cursor is called the result table for the cursor, or simply, the result table.

### scrollable cursor

A scrollable cursor is one that permits arbitrary navigation through the rows of the result table for a cursor. In contrast, a non-scrollable cursor is one that is forward-moving only and permits the rows of the result table to be fetched once and once only, in a predetermined order.

DRDA support for scrollable cursors is based on the ANSI SQL99 standard, though it includes some extensions specific to DRDA.

## B.2 Attributes

### B.2.1 Scrollability Attribute

Whether the cursor is scrollable or not is specified by the application at the time the cursor is defined. The application server returns the cursor's scrollability attribute with the OPNQRYSRM when the cursor is opened at the application server.

For a result set defined as scrollable by a stored procedure, the defined scrollability attribute will be overridden by the application server if the application requester is at an SQLAM level less than 7. The application server does not return any scrolling attributes for the result set. Thus, to the application requester and to the calling application, the result set is non-scrollable. This behavior allows an application on a downlevel application requester to use stored procedures whose result set definitions they do not necessarily know or control.

For more information, see [Section B.9](#) (on page 714). See also the QRYATTSCR term in the DDM Reference.

## B.2.2 Sensitivity Attribute

SQLAM Level 8 and above supports five kinds sensitivity for scrollable cursors:

- Insensitive

This specifies that the cursor does not have sensitivity to inserts, updates, or deletes made to the rows underlying the result table once the result table for the cursor is materialized. Consequently, the size of the result table, the order of the rows, and the values for each row do not change after the cursor is opened. Additionally, the cursor is read-only.

- Sensitive Static

This specifies that the cursor has sensitivity to changes made to the database after the result table for the cursor is materialized, and that the result table has a static size and ordering.

- The cursor is always immediately sensitive to changes made using the cursor (that is, positioned updates and deletes using the same cursor). Whether changes made outside the cursor are visible to the cursor depends on the type of FETCH that is used (see [Section B.3](#) (on page 709)).
- The size of the result table does not grow after the cursor is opened and the rows are materialized. The order of the rows is established as the result table is materialized.

To present a static size and static ordering for the result table, the relational database may return a hole to the application that fetches an updated or deleted row in the result table. A hole in the result table occurs when there is a difference between the result table and the underlying base table. No data can be fetched from a hole, and the hole is manifested in the QRYDTA as a row consisting of a non-null SQLCARD and a null data group.

When the current value of a row no longer satisfies the *select-statement* or *statement-name*, that row is visible in the cursor as an “update hole”, where the SQLCARD has a warning SQLSTATE of 02502.

When a row of the result table is deleted from the underlying base table, the row is visible in the cursor as a “delete hole”, where the SQLCARD has a warning SQLSTATE of 02502.

- Sensitive Dynamic

This specifies that the cursor has sensitivity to changes made to the database after the result table for the cursor is materialized and that the result table has a dynamic size and ordering.

- The cursor is always immediately sensitive to changes made using the cursor (that is, positioned updates and deletes using the same cursor) and is always sensitive to changes made outside the cursor.
- The size of the result table may change after the cursor is opened as rows are inserted into the underlying table, and the order of the rows may change. Holes do not occur for a sensitive dynamic cursor.

- Partially Sensitive Static

The protocol rules for this type of cursor are exactly the same as for sensitive static. The attributes are the same, with the exception that the query in question has subqueries, and that some of the queries that make up the statement are implemented as insensitive while other queries are sensitive. This could happen, for example, if the outer query is insensitive but one or more of the subqueries were implemented as sensitive.

- Partially Sensitive Dynamic

The protocol rules for this type of cursor are exactly the same as for sensitive dynamic. The attributes are the same, with the exception that the query in question has subqueries, and that some of the queries that make up the statement are implemented as insensitive while other queries are sensitive.

In the following discussion, references to *sensitive* cursors should be interpreted to include *partially sensitive* cursors as well, since they really are a subset and the same rules apply.

The sensitivity of a scrollable cursor is specified by the application when the cursor is defined. This is known as the defined sensitivity of the scrollable cursor. The relational database engine determines the effective sensitivity of the scrollable cursor when opening the cursor. The effective sensitivity may or may not be the same as the defined sensitivity. For example, a cursor defined as sensitive dynamic may actually have an effective sensitivity of insensitive because of the way in which the cursor is materialized.

The application server returns the cursor's effective sensitivity attribute with the OPNQRYRM when the cursor is opened at the application server.

See also the QRYATTSENS term in the DDM Reference.

### B.2.3 Updatability Attribute

A scrollable cursor may or may not be updatable. If it is updatable, then fetched rows in the result table can be modified and can cause the base table of the cursor to be modified as well. Whether a scrollable cursor is updatable is not solely dependent on how the cursor is defined, but can be influenced by both the SELECT statement and by the cursor sensitivity option specified when the cursor is defined. Thus, the defined and effective updatability may be different.

The application server returns the cursor's effective updatability attribute with the OPNQRYRM when the cursor is opened at the application server. The value returned indicates whether the cursor is updatable and, if updatable, whether the type of updatability (for example, if updatable via DELETE statements only or via both DELETE and UPDATE statements).

See also the QRYATTUPD term in the DDM Reference.

## B.3 Operations

Because a scrollable cursor allows for both forward and backward movement through the result table, a FETCH operation may include information on how to navigate through the result table. SQLAM Level 7 and above provides for the following navigational control on the CNTQRY command.

- Scroll Orientation

This specifies the desired placement of the cursor as part of a FETCH statement. Scroll orientation may be used in conjunction with a row number or may be used independently. For example, a scroll orientation of AFTER places the cursor after the last row in the result table.

- Row Number

This specifies the desired row in the result table, when used in conjunction with either a RELATIVE or ABSOLUTE scroll orientation, and places the cursor at a specific row number, either relative to the current cursor position or with respect to the beginning or end of the result table.

- Return Data Option

This specifies whether the FETCH returns data. If no data is returned, the FETCH is called a positioning FETCH and results only in changing the cursor position according to the scroll orientation and row number, if specified.

For a sensitive dynamic scrollable cursor, all FETCH requests are sensitive to changes made by this cursor, to changes made by other cursors in the same application process, and to committed changes made by other application processes.

For a sensitive static scrollable cursor, a given FETCH request is always immediately sensitive to changes made through the cursor. To control whether the FETCH request must also reflect the committed updates and deletes made by all other processes, the application may also include the desired sensitivity to others' changes on the FETCH request. SQLAM Level 7 and above provide for the following row sensitivity control on the CNTQRY command.

- Sensitive

This indicates that the FETCH is to be sensitive to all updates and deletes made by this cursor and by committed updates and deletes by all other application processes.

- Insensitive

This indicates that the FETCH is not to be sensitive to updates and deletes made outside this cursor. However, it is sensitive to all explicit updates and deletes made by this cursor.

DRDA does not validate the row sensitivity specified on a CNTQRY command. The relational database determines whether a row sensitivity is valid for a given FETCH request, given the effective sensitivity attribute of the cursor.

See also the QRYSCRORN, QRYROWNBR, and QRYROWSNS terms in the DDM Reference.

## B.4 Choice of Query Protocol

SQLAM Level 7 and above supports scrollable cursors via both the Fixed Row Query Protocol and the Limited Block Query Protocol. An application server at SQLAM Level 7 or higher selects the Fixed Row Query Protocol for a scrollable cursor if the cursor is updatable, if the scrollable cursor is required to be sensitive to changes with each FETCH request, or if the application requester forced the choice of the Fixed Row Query Protocol. Otherwise, the application server may select the Limited Block Query Protocol if the cursor does not perform updates, if the scrollable cursor is not required to be sensitive to changes with each FETCH request, and if the application requester did not force the choice of the Fixed Row Query Protocol. In particular, the application server must select the Fixed Row Query Protocol for sensitive dynamic scrollable cursors.

See the QRYBLKCTL term in the DDM Reference for more information.

For a scrollable cursor using the Fixed Row Query Protocol, the cursor is accessed by the application requester in a non-scrollable fashion if no scrolling parameters are specified on the OPNQRY and CNTQRY commands. Such an OPNQRY command retrieves no rows and each CNTQRY results in a FETCH NEXT operation at the application server that returns just one row (as for standard Fixed Row Query Protocol). On the other hand, the scrollable cursor is accessed in a scrollable fashion when the application requester specifies one or more of the scrolling operations that control the navigation and sensitivity of the FETCH request. Each such CNTQRY command returns only the one row that satisfies the navigational and sensitivity requirements of the FETCH request.

For a scrollable cursor using the Limited Block Query Protocol, the cursor is accessed by the

application requester in a non-scrollable fashion if no scrolling parameters are specified on the OPNQRY and CNTQRY commands. Such an OPNQRY command retrieves an architecturally-fixed number of rows (64, as defined by QP4<sup>75</sup>; see Section 7.22.3 (on page 484)) while each such CNTQRY fetches rows according to the standard Limited Block Query Protocol. On the other hand, the scrollable cursor may be accessed in a scrollable fashion only when the QRYROWSET parameter is specified on the CNTQRY command. An application requester may not mix scrollable and non-scrollable behavior for such a cursor.

## B.5 DRDA Rowsets

DRDA can block the rows returned by a specified number of single row fetches into a single DRDA rowset to be returned to the application. Because the rows are retrieved by a single command, the operation is more network-efficient. The number of rows for the DRDA rowset is specified by the *qryrowset* parameter on OPNQRY, CNTQRY, or EXCSQLSTT for a non-rowset cursor.

The requester specifies the *qryrowset* parameter (say, it has a value of *S*) on the OPNQRY, CNTQRY, or EXCSQLSTT command when it wishes to retrieve a DRDA rowset, consisting of no more than *S* rows of the result table for the cursor or for the result sets returned by a stored procedure call. A requester retrieves a DRDA rowset of rows either when accessing a cursor in a scrollable manner in order to get the performance benefits of query blocking while taking into account the requirements of the scrolling behavior, or when accessing a read-only non-scrollable cursor in order to limit the number of rows to be returned with the command when block fetching is in effect.

The *qryrowset* parameter may be specified for non-dynamic scrollable cursors and may be specified for either the Fixed Row Query Protocol or the Limited Block Query Protocol.

The *qryrowset* parameter may also be specified for non-scrollable and non-rowset cursors using the Limited Block Query Protocol.

The query blocks containing the DRDA rowset adhere to the rules for Limited Block Query Protocol, even though the query protocol in effect for the cursor is Fixed Row Protocol. While a Fixed Row Protocol cursor may access the cursor in a scrollable manner with or without the *qryrowset* parameter, the *qryrowset* parameter is required for Limited Block cursors to indicate that the cursor will be used in a scrollable manner or for Fixed Row rowset cursors.

When a *qryrowset* parameter is specified on the OPNQRY command, the first row in the DRDA rowset returned by the application server consists of the first row in the result table, followed by the next *S*-1 rows in sequence (FETCH NEXT) in the result table. When a *qryrowset* parameter is specified on a CNTQRY command, the first row in the DRDA rowset consists of the row identified by the navigational parameters on the CNTQRY command, followed by the next *S*-1 rows in sequence (FETCH NEXT) in the result table.

The DRDA rowset is said to be “completed” when the requested number of rows (*S*) are fetched or when a FETCH request at the application server results in a negative SQLSTATE or an SQLSTATE of 02000, or when the CNTQRY command identifies a positioning FETCH. The intent of a positioning FETCH is to change the position of the cursor but explicitly does not request the return of return data (for example, FETCH AFTER or a FETCH request that does not have a

---

75. This value is an arbitrarily-chosen architectural constant that limits the number of rows returned on the OPNQRY request for a scrollable cursor. It allows the application server to create an implicit rowset of a known size in case the application requester decides to use the cursor in a scrollable fashion with subsequent CNTQRY requests. It can be overridden by the application requester by means of the QRYROWSET parameter on the OPNQRY request for scrollable cursors. It has no effect on subsequent CNTQRY requests if the application requester does not access the cursor in a scrollable manner.

fetch target list). Otherwise, the DRDA rowset is said to be “incomplete” and it is the application requester’s responsibility to dispose of the incomplete DRDA rowset by either completing the DRDA rowset or resetting the DRDA rowset. Refer to Rule QP4 in [Section 7.22.3](#) for the responsibilities of the application requester and application server with respect to incomplete DRDA rowsets.

The effect of a command that returns a DRDA rowset is equivalent to performing the specified number of single-row fetches across the network, but since they are retrieved by a single command, the operation is more network-efficient. By fetching a DRDA rowset, the application requester gets the benefit of blockfetching for both updatable and read-only scrollable cursors. On the other hand, for non-scrollable cursors, the application requester improves network performance by allowing greater control of the number of rows that can be returned with each command of a block-fetched cursor. DRDA does not define the manner in which the application requester determines the DRDA rowset value to be specified.

See also the OPNQRY, CNTQRY, and QRYROWSET terms in the DDM Reference for more information.

## B.6 Cursor Position Management

When an OPNQRY or CNTQRY command with a QRYROWSET parameter is executed for a scrollable cursor, the application server fetches rows in advance of the application’s fetching those rows. Thus, the cursor position known to the application may be different from the cursor position known to the application server. It is the responsibility of the application requester to ensure that the application requests that depend on cursor position are performed correctly in one of two ways:

1. Forcing equivalence of the cursor position.

The application requester ensures that the application’s cursor position is the same as the application server’s cursor position. This occurs naturally when each application FETCH request causes only one FETCH operation at the application server. If the application requester asks for more than one row by means of the QRYROWSET parameter, then the application requester must ensure that the application fetches all rows returned by the application server before it allows the application to perform any other operations on the cursor.

The application requester may safely choose this method both for updatable cursors where the application requires correct knowledge of the server’s cursor position to make updates through the cursor and for read-only cursors.

If the application requester chooses this approach and also specifies a QRYROWSET parameter, then it would probably want to complete any incomplete rowsets it receives from the application server since it will be returning all retrieved rows to the application.

2. Mapping differences in the cursor position.

If the application requester does not force the equivalence of the cursor position, then it must map the differences in the application’s and the target server’s cursor positions so that the proper row is fetched or updated. For example, suppose an application issues a fetch request that causes the application requester to send a CNTQRY command to the application server that requests a rowset. Say, a complete rowset is returned to the application requester consisting of row  $N$  to row  $N+S$ . Then, the current cursor position on the application server is  $N+S$ , but after the application requester returns row  $N$  to the application, the current cursor position at the application is  $N$ . An UPDATE request through the cursor at this point would reference the wrong row.

Similarly, a `FETCH RELATIVE R` request, where  $R > S$ , that is sent as a `CNTQRY` with a `RELATIVE R` specification would fetch row  $N+S+R$ . To accurately reflect the application's desires, the application requester could instead send a `CNTQRY` request specifying `ABSOLUTE N+R`. Note, in this example, that the application requester discards the unread rows in the previous rowset to fetch a row not in the received `QRYDTAs`.

The application requester may not choose this method for updatable cursors since this method does not ensure updates through the cursor are positioned correctly relative to the server's cursor position. The application requester may safely choose this method only for read-only cursors.

If the application requester chooses this approach and also specifies a `QRYROWSET` parameter, then it cannot ensure that all retrieved rows will be returned to the application. Thus, some retrieved rows may be discarded without ever being passed to the application. If a row is to be discarded, then the application requester is responsible for discarding all the data associated with the row, including any unreceived `EXTDTA DSSs` containing `LOB` data. Note that since the `RTNEXTDTA` option is required to be `RTNEXTALL` in this case, there is no pending `LOB` data at the server. Generally speaking, the application requester would probably want to reset any incomplete rowsets in order to pass the navigational information required to `FETCH` the row desired by the application.

Since sensitive dynamic scrollable cursors have dynamically changing membership and ordering within the result table, it is not possible to accurately derive absolute row numbers as updates are made to the cursor. Thus, it is not possible to manage cursor positions as described above when a rowset is fetched. If a `QRYROWSET` value is specified on an `OPNQRY` for a sensitive dynamic scrollable cursor, the application server ignores it; if it is specified on a `CNTQRY` command, the application server rejects the command with a `PRCCNVRM`.

When an application requester accesses a cursor without specifying any `QRYROWSET` values on the `OPNQRY` and `CNTQRY` commands, it does not need to manage the cursor position since it either intends to access the cursor in a non-scrollable manner or the application requester will never generate any cursor position differences between the application and the relational database at the application server.

## B.7 Cursor Position Rules

When the `QRYROWSET` parameter is specified, cursor position management is required. In the case of the rows returned with the `OPNQRY` command for a scrollable cursor using the Limited Block Query Protocol, where no rowset is specified, the rows returned are considered to be an implicit rowset if the cursor will be used in a scrollable manner (that is, each subsequent `CNTQRY` command includes a `QRYROWSET` value). Cursor position management is required for the implicit rowset in this case as well. To accommodate the requirements of cursor position management, the application requester must always be able to determine the cursor position of each row returned in a rowset.

Thus, the following cursor position rules apply:

- After a scrollable cursor is opened, the cursor position is before the first row of the result table. Thus, for example, the first row in an implicit or explicit rowset returned with an `OPNQRY` command has cursor position 1.

- When a fetch request navigating forward needs to fetch beyond the end of the result table, the application server returns an SQLSTATE of 02000 and the cursor position is set after the last row of the result table.
- When a fetch request navigating backwards needs to fetch beyond the start of the result table, the application server returns an SQLSTATE of 02000 and the cursor position is set before the first row of the result table.
- When the current cursor position is not on a row of the result table (the row may be a hole), a fetch request for the current row causes the application server to return an SQLSTATE of 02000 and the cursor position is not changed.
- When a query scroll orientation of QRYSCRBEF or QRYSCRAFT is executed, the application server returns an SQLSTATE of 00000.

## B.8 Cursor Disposition

A scrollable cursor is not terminated by an SQLSTATE of 02000. A CLSQRY command is required to close the cursor, unless a terminating error occurs while the cursor is being accessed. If a terminating error occurs, the ENDQRYRM is returned according to the rules for the query protocol in effect.

## B.9 Scrolling for Stored Procedure Result Sets

A stored procedure may define a result set to be scrollable. The OPNQRYRM for the result set contains the cursor's scrollability, sensitivity, and updatability attributes. Both the stored procedure and the calling application may scroll all the rows in the result table.

If the stored procedure accesses the cursor for the result set, the cursor position is unchanged when the stored procedure completes. Because of the requirements of the application requester for managing the cursor position, DRDA imposes a restriction: if the result set is not positioned before the first row of the result table when the stored procedure completes, then the stored procedure call statement will fail with an SQLSTATE of 560B1. This ensures that the first row in the implicit rowset returned is row 1.

As described in [Section B.2.1](#) (on page 707), the application server reverts the scrollable result set to non-scrollable if the application is calling the stored procedure from a downlevel application requester. In such a case, the stored procedure may access the cursor in a scrollable manner but the application is restricted to non-scrolling access to the result set. Such a result set is not restricted to being positioned before the first row of the result table when the stored procedure completes.

## B.10 Downlevel Requesters

If a cursor is scrollable, then the application server first checks that the application requester supports scrollable cursors before returning an OPNQRYM. If the application requester is not at SQLAM Level 7 or higher, then the server fails the OPNQRY command with a SQLSTATE of 560B2. The cursor, which has been successfully opened by the relational database is closed.

If any result sets returned by a stored procedure are scrollable result sets, then the application server also checks that the application requester supports scrollable cursors before responding to the EXCSQLSTT. If the application requester is not at SQLAM Level 7 or higher, then the server acts according to whether it is the target server or an intermediate server. The target application server reverts all scrollable result sets to non-scrolling result sets while an intermediate server fails the stored procedure call with an SQLSTATE of 560B3. If the stored procedure call is failed, all result sets are closed.

## B.11 Intermediate Data Server Processing

If a QRYROWSET value is specified by an application requester, each intermediate data server is responsible for returning the requisite number of rows (pushing the rows) up to the number determined by the extra query block limits. Only the application requester site is responsible for managing the differences in cursor position. Each intermediate data server allows the upstream site to pull the correct number of rows, but manages the QRYROWSET value as follows to ensure the correct number of rows is pushed.

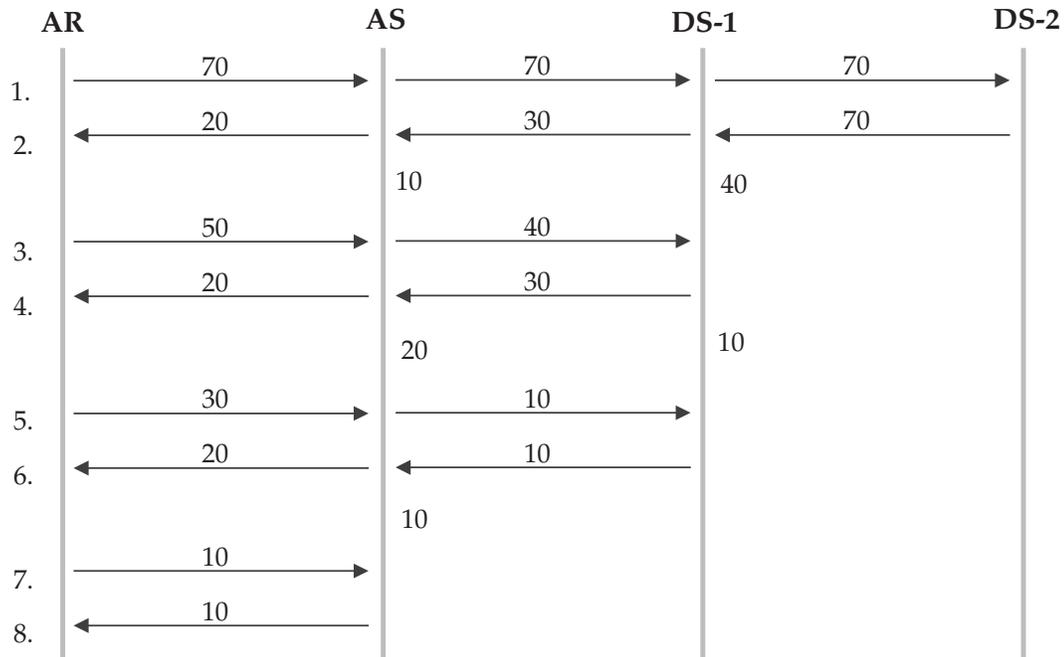
### B.11.1 Example: qryrstblk Not Specified

Suppose a CNTQRY QRYROWSET value of  $S$  is received at an intermediate data server. Suppose that the downstream site pushes  $S_1 < S$  number of rows because of extra query block limits. Finally, suppose this data server site only pushes  $S_2 < S_1$  number of rows to the upstream site because of its extra query block limits. If a subsequent CNTQRY request is received with a QRYROWSET value of  $T$  (and the *qryrstblk* parameter is not specified), then the data server must pull the  $S_1 - S_2$  number of pending rows before forwarding the new rowset value of  $T - (S_1 - S_2)$  to the downstream site.

In [Figure B-1](#) (on page 716),  $S=70$ . In the typical scenario, it is expected that all 70 rows will be returned in one query block through all the intermediate servers, so that only one CNTQRY is required for a complete rowset. This is an example to show the dynamics of the flow in an extreme case.

1. The CNTQRY request is passed through the intermediate sites to the target server,  $DS-2$ .
2. In this case, the target server can return all 70 requested rows to the upstream intermediate server,  $DS-1$ . The query block size and extra query block limits allow  $DS-1$  to return only 30 of the rows to the application server. This means that 40 rows are pending at  $DS-1$ . The application server, in turn, is restricted its extra query block limit to returning only 20 rows to the application requester. This means that 10 rows are pending at the application server.
3. The application requester returns all 20 rows to the application and now requests another CNTQRY for the remaining rows (50). The application server consumes the 10 rows pending and sends a CNTQRY for the 40 rows required to complete the rowset of size 50.  $DS-1$  uses the pending 40 rows to satisfy this request.

4. Again, *DS-1* can only return 30 rows to the application server. This leaves 10 rows pending at *DS-1*. The application server, in turn can only return 20 rows of the 40 rows obtained from *DS-1*. This leave 20 rows pending at the application server.
5. The application requester returns the 20 rows to the application, for a total of 40 rows. The application requester requests the remaining 30 rows from the application server. The application server consumes the 20 pending rows and requests 10 more rows from *DS-1*.
6. Those 10 rows are already pending and are returned by *DS-1*. The application server can only return 20 rows to the application requester, which leaves 10 rows pending at the application server.
7. Again, the 20 rows are returned to the application, for a total of 60 rows. The application requester requests the remaining 10 rows from the application server.
8. Those 10 rows are already pending and are returned by the application server. The application requester returns the 10 rows to the application and that completes the rowset.



**Figure B-1** Scrollable Cursors: Example with `qyrstblk` Not Specified

### B.11.2 Example: `qyrstblk` Set to TRUE

Suppose a `CNTQRY QRYROWSET` value of  $S$  is received at an intermediate data server. Suppose that the downstream site pushes  $S1 < S$  number of rows because of extra query block limits. Finally, suppose this data server site only pushes  $S2 < S1$  number of rows to the upstream site because of its extra query block limits. If a subsequent `CNTQRY` request is received with a `QRYROWSET` value of  $T$  and a `qyrstblk` parameter of `TRUE`, then the data server discards the pending rows before forwarding the `CNTQRY` with the `QRYROWSET` value of  $T$  and `qyrstblk` of `TRUE`.

In [Figure B-2](#) (on page 717),  $S=64$ . In the typical scenario, it is expected that all 64 rows will be

returned in one query block through all the intermediate servers, so that only one CNTQRY is required for a complete rowset. This is an example to show the dynamics of the flow in an extreme case.

1. The CNTQRY request is passed through the intermediate sites to the target server, *DS-2*.
2. In this case, the target server can return all 64 requested rows to the upstream intermediate server, *DS-1*. The query block size and extra query block limits allow *DS-1* to return only 24 of the rows to the application server. This means that 40 rows are pending at *DS-1*. The application server, in turn, is restricted by its extra query block limit to returning only 14 rows to the application requester. This means that 10 rows are pending at the application server.
3. The application requester returns some or all of the 14 rows to the application and now determines that it needs to start another rowset, using the navigational information from the application (*rownbr*). The application requester ensures that any pending rows are discarded, any unreceived extra query blocks, and any unreceived LOB EXTDTA objects. The application requester now sends another CNTQRY to the application requester specifying a rowset size of 64, setting *qryblkrst* to TRUE, and passing the row number. The application server discards its pending rows, including pending extra query blocks and pending EXTDTA objects. The application server forwards the request to *DS-1* which also discards unreceived and pending data from the previous rowset. *DS-1* then forwards the request to *DS-2* which also discards unreceived and pending data from the previous rowset. *DS-2* then processes the CNTQRY request, issuing the first fetch according to the navigational parameters on the command. The processing repeats as above.

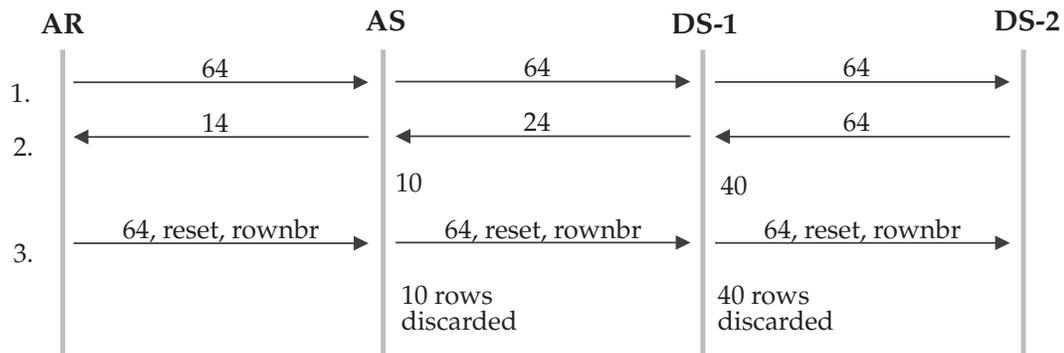


Figure B-2 Scrollable Cursors: Example with *qryrstblk* Not Specified



## C.1 Rowset Cursors

**Note:** The support in DRDA Level 5 supersedes the multi-row fetch defined in DRDA Level 2.

A rowset cursor is a cursor defined such that more than one row can be returned for a single fetch statement called multi-row fetch. Rowset cursors support multi-row fetch; rows are fetched as an atomic operation allowing rowset positioning. With a rowset cursor, the cursor is positioned on more than one row. Specifically, the cursor is positioned on the set of rows fetched. Each row of the cursor position for a rowset cursor can be referenced in subsequent positioned delete and update statements. A fetch statement for a rowset cursor specifies a rowset-positioned fetch orientation clause, and can indicate the desired number of rows for the SQL rowset (the set of rows returned by a fetch against a rowset cursor).

Rowset cursors ignore the *qryrowset* parameter on the OPNQRY and EXCSQLSTT command. The *qryrowset* parameter does not have to be specified on the CNTQRY for a rowset cursor. If a *qryrowset* value is not specified on the CNTQRY when fetching from a rowset cursor, then a single-row fetch is being performed. Rowset cursors must use the Fixed Row Query Protocol. When processing a rowset cursor, flexible blocking is used. See Block Formats (BF Rules) in [Section 7.22.1.1](#) for detail on flexible blocking.

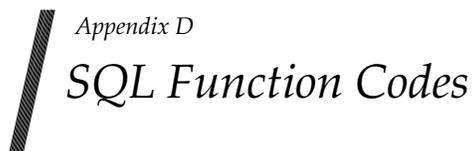
## C.2 SQL Rowsets

An SQL rowset contains the answer set (consisting of one or more rows) of a multi-row fetch against a rowset cursor as defined above. Only one SQL rowset can be returned as the result of a multi-row fetch. For multi-row fetch, the requester must provide a statement-level SQLCA to the application. For rowset cursors, the server returns the statement-level SQLCA for the SQL rowset for each QRYDTA that contains the row or rows in the SQL rowset. The row-level SQLCAs in the QRYDTA are set to null because the fetch for a rowset cursor is an atomic operation and only one SQLCA is returned to the application.

In contrast, a non-rowset cursor is a cursor defined such that the cursor is positioned on a single row, and all cursor operations occur on a single row only. Although only single row fetches can be performed on non-rowset cursors, a DRDA rowset can be returned for a non-rowset cursor. See [Appendix B](#) for more details on non-rowset cursors and DRDA rowsets.

SQL rowset processing offers performance advantages similar to DRDA rowset processing in terms of network performance (as described in [Appendix B](#)), but also has the added benefit of allowing subsequent operations to operate on the SQL rowset and take advantage of rowset positioning.





Appendix D  
SQL Function Codes

Below are the SQL statement codes as defined in Section 18.1, <get diagnostics statement> of ISO/IEC 9075-5:1999:

ALLOCATE CURSOR	1 (one)
ALLOCATE DESCRIPTOR	2
ALTER DOMAIN	3
ALTER ROUTINE	17
ALTER TYPE	60
ALTER TABLE	4
CALL	7
CLOSE CURSOR	9
COMMIT WORK	11
CONNECT	13
CREATE ASSERTION	6
CREATE CHARACTER SET	8
CREATE COLLATION	10
CREATE DOMAIN	23
DEALLOCATE DESCRIPTOR	15
DEALLOCATE PREPARE	16
DECLARE CURSOR	101
DELETE CURSOR	18
DELETE WHERE	19
DESCRIBE	20
DISCONNECT	22
DROP ASSERTION	24
DROP CHARACTER SET	25
DROP COLLATION	26
DROP TYPE	35
DROP DOMAIN	27
DROP ROLE	29
DROP ROUTINE	30
DROP SCHEMA	31
DROP TABLE	32
DROP TRANSFORM	116
DROP TRANSLATION	33
DROP TRIGGER	34
DROP ORDERING	115
DROP VIEW	36
DYNAMIC CLOSE	37
DYNAMIC DELETE CURSOR	54
DYNAMIC DELETE CURSOR	38
DYNAMIC FETCH	39
DYNAMIC OPEN	40
DYNAMIC UPDATE CURSOR	42
DYNAMIC UPDATE CURSOR	55
EXECUTE IMMEDIATE	43
EXECUTE	44

FETCH	45
FREE LOCATOR	98
GET DESCRIPTOR	47
HOLD LOCATOR	99
GRANT	48
GRANT ROLE	49
INSERT	50
OPEN	53
PREPARE	56
RELEASE SAVEPOINT	57
RETURN	58
REVOKE	59
SELECT	21
SELECT	41
SELECT CURSOR	85
SET CATALOG	66
SET DESCRIPTOR	70
SET CURRENT_PATH	69
SET NAMES	72
SET SCHEMA	74
SET TRANSFORM GROUP	118

For server-specific function codes, refer to the server's documentation if not defined in the above list.

## Failover Overview

DRDA provides two flavors of failover support, depending on whether the RDB is replicated or not. When the RDB is not replicated, failover support may be provided through multiple servers. When the RDB is replicated, DRDA allows the requester to reroute connections to another server where the replica resides.

### Single RDB with Multiple Servers

This is a setup where a collection of servers access a single RDB. A requester can connect to any server within the collection and perform SQL operations on the same data concurrently. With this setup, all servers operate in parallel to provide access to a single RDB for workload balancing as well as for high availability. Connectivity and load information on each of the servers are returned in the Server List (SRVLST) on the ACCRDBRM reply message at connect time. For TCP/IP, all such servers share a common host name, even though they all have different IP addresses. If one server becomes inaccessible, the requester can attempt to access the RDB based on the connectivity information for another server in the collection as returned in the SRVLST.

Resolution is required for any transaction that is indoubt after a communications failure. For SYNCPTMGR-protected connections, the source SYNCPTMGR attempts to access the target SYNCPTMGR using the network address contained in the SYNCLOG. Over TCP/IP, if the target SYNCPTMGR is no longer accessible using the IP address contained in the SYNCLOG, the source SYNCPTMGR can attempt to access the target SYNCPTMGR by resolving to a new IP address from the host name provided in the SYNCLOG. For XAMGR-protected connections, once a connection to the RDB has been reestablished through any server in the SRVLST collection, the source XAMGR can retrieve a list of XIDs for prepared or heuristically completed transactions from the target XAMGR. Such indoubt transactions are then resolved by the source XAMGR.

### Replicated RDB

In this environment, an RDB is replicated at one or more alternate server locations, each having its own IP address and host name. DRDA does not architect how the RDB is replicated. However, the replication must include the log, complete with information on any indoubt transactions. At connect time, the Server List (SRVLST) on the ACCRDBRM reply message provides connectivity information on alternate server locations for the RDB. The replicated RDB at each alternate server location is unusable unless the RDB has failed over to that replica. How RDB failover is effected is beyond the scope of DRDA and is therefore unarchitected.

If a database connection fails and the RDB has successfully failed over to a replica thereof at an alternate server location, the requester will be able to reconnect to the RDB at the new location using connectivity information which was returned previously in the SRVLST.

Since the log of the RDB must be replicated at an alternate server, any existing indoubt transaction must also exist at the replicated RDB. However, resynchronization of an indoubt transaction on a SYNCPTMGR-protected connection over TCP/IP may not be possible at the alternate location because the SYNCLOG connectivity information (both IP address and host name) may no longer be valid. Only after the RDB has failed back to the original server can resynchronization of the indoubt transaction occur. For XAMGR-protected connections, once a connection to the RDB has been reestablished through an alternate server, the source XAMGR can retrieve a list of XIDs for prepared or heuristically completed transactions from the target

XAMGR. Such indoubt transactions are then resolved by the source XAMGR.

**Restoration of Execution Environment After Failover**

When the requester accesses an RDB in either failover environment, whenever a special register setting gets updated, the requester can request the server to return one or more SQLSTT reply data objects, each containing an SQL SET statement for any special register that has had its setting modified on the current connection. The requester can cache these statements for use later when it needs to reestablish a connection to the RDB. Once the connection is reestablished, the requester should flow an EXCSQLSET command to the server along with the SQL SET statements which were returned earlier should it wish to restore the execution environment for the application.

# 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** 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 partition**

A part of the database that consists of its own user data, indexes, configuration files, and transaction logs. Sometimes called a node or database node. See partitioned database.

**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 codepoints 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 Data Format**

A support that allows FD:OCA Generalized String data in an answer set to be returned in a representation that is determined by the server at the time when the data is retrieved based on its actual length.

**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.

**partitioned database**

A database with two or more database partitions. Data in user tables can be located in one or more database partitions. When a table is on multiple partitions, some of its rows are stored in one partition and others are stored in other partitions. See database partition.

**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.

**progressive reference**

An 8-byte data reference through which the actual value bytes can be retrieved using the DDM command GETNXTCHK. It itself is returned in Mode X'03' of the Dynamic Data Format; see Data Format (DF Rules) in [Section 7.8](#) for more details. The life of a progressive reference is tied to its originating cursor; i.e., if the cursor is closed implicitly or explicitly, the progressive reference will also be freed. The name "progressive" indicates that the data

returned using such reference is always progressive or sequential.

**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.

**result set component**

A set of reply objects returned in response to an EXCSQLSTT command for a stored procedure call that returns one or more result sets. As a set, these objects identify a result set returned by the stored procedure, describe the columns in the answer set returned by the result set, and may also contain some or all of the data returned by the result set. The objects that make up a result set component as well as the ordering of the objects in a result set component are given in the Query Data Transfer Protocols rules (see QP5).

**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 codepoints 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.

**summary component**

A set of reply objects returned in response to an EXCSQLSTT command for a stored procedure call that returns one or more result sets. As a set, these objects give the completion status of the stored procedure call, return the parameters, if any, and give information about the result sets. The objects that make up the summary component as well as the ordering of the objects in the summary component are given in the Query Data Transfer Protocols rules (see QP5).

**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 component**

A set of reply objects returned in response to an EXCSQLSTT command for a stored procedure call that returns one or more result sets. As a set these objects give the transaction status of the stored procedure call. If the transaction state has not changed as a result of the execution of the stored procedure call, then the transaction component may not be returned. The objects that make up the transaction component are given in the Query Data Transfer Protocols rules (QP5).

**Note:** A stored procedure call that does not return result sets may also return these objects, but as a set they are not referred to as a transaction component.

**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. User IDs 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

ABEND .....	212, 543, 596, 604	array input.....	193
ABNUOWRM .....	199, 233, 452, 503	AS.....	725
AC rule.....	427	assigning LIDs to O triplets .....	393
access path.....	419	assigning override.....	393
access relational database.....	75	asynchronous wait .....	583
accounting .....	514	atomic chaining.....	16
accounting information .....	54, 575, 580, 614	atomic chaining rule .....	427
ACCRDB.....	84, 277, 397, 615	ATTACH .....	582
ACCRDBRM .....	95, 263, 277, 503, 589	authenticated conversation.....	54
ACCSEC.....	97, 585, 615	authentication .....	516, 575, 580, 614
ACCSECRD.....	97, 585, 615	authorization.....	575
actions .....	550	BACKOUT.....	579, 602
additional SV.....	552	base and option sets .....	248
agent.....	71, 81-82, 91	base set functions.....	577
AGNPRM alert .....	549	base set of LU 6.2 .....	577
AGNPRMRM .....	465, 503	basic conversation .....	605
alert.....	545, 725	basic conversation verb .....	578
alert and supporting data.....	604	basic FD:OCA object .....	248
alert at application requester .....	548	Begin Bind option.....	12
alert descriptions .....	550	BF rule .....	474
alert example.....	566	BGNBND .....	75, 127, 129, 256, 590, 616
alert generation.....	545	BGNBNDRM.....	129, 503
alert implementation basics .....	545	bind .....	57-58, 419, 725
alert model mapping.....	546	bind copy operation .....	132
alert models.....	548	bind deploy operation .....	136
alert structures .....	545	bind flows.....	127, 131, 590, 616
alert to reply message mapping.....	546	bind option values.....	30
alert types .....	550	bind options .....	17, 33, 129
alias.....	424	BLKERR alert .....	553
ALLOCATE .....	416, 577, 580	block chaining rule.....	478
alphabetic extender .....	417	block format rule .....	474
API.....	46, 725	block size rule .....	477
APPC.....	575	blocking.....	473
APPCMNI.....	580	blocking protocol error .....	548, 553
APPCMNT .....	596	blocking rule .....	473
application programming interface.....	46	BNDOPT .....	129, 144, 456
Application Programming Interface (API).....	725	BNDSQLSTT .....	75, 127, 129, 214, 256, 590, 616
application requester .....	49, 53, 70, 88	BNDSTTASM .....	455
application requester (AR).....	725	boolean.....	376
application server.....	49, 53, 70, 88	BQUAL.....	235
application server (AS).....	725	BS rule .....	477
application services.....	613	buffered insert.....	15
application support protocol .....	49, 53, 725	CA rule.....	429
APPSRCCD .....	589	causes .....	550
APPSRCCR.....	589	CCSID.....	37, 277, 335, 386, 436, 450, 506, 725
AR.....	725	CCSID manager .....	77, 81
ARM correlator .....	20	CCSIDDBC .....	429
		CCSIDMBC.....	429

- CCSIDSBC .....429  
 CCSIDXML.....429  
 CD rule.....431  
 CDRA .....45, 84, 335, 725  
 CF rule.....431  
 CH rule.....478  
 chaining rule .....478  
 chaining violation.....548  
 character set restriction.....506  
 CHNVIO alert.....554  
 CLSQRY .....59, 75, 148, 153, 161, 256  
 CMDATHRM.....503  
 CMDCHK alert.....555  
 CMDCHKRM.....465, 503  
 CMDNSPRM.....453, 503  
 CMDVLT alert.....556  
 CMDVLTRM.....230-231, 433, 465, 503  
 CMMRQSRM.....220, 225, 227, 230-231, 433, 503  
 CMNAPP .....70, 81, 575  
 CMNMGR .....513  
 CMNSYNCPT .....71, 81, 575  
 CMNTCPIP manager .....71, 611  
 CNTQRY .....59, 75, 148, 152-153, 253-254, 256  
 Coded Character Set Identifier (CCSID) .....725  
 codepoint .....81, 84, 253  
 CODPNT.....84  
 CODPNTDR.....84  
 coexistence .....217-218, 221  
 COLLECTION .....418-419  
 collection ID .....21, 131  
 command and reply flow .....88  
 command data object.....74  
 command execution example.....401  
 command source identifier .....19, 421  
 command/descriptor relationship .....256  
 commit .....61  
 COMMIT .....212, 218-219, 222, 225-226, 229  
 commit .....232  
 commit and rollback.....214-215  
 commit and rollback processing rule .....432  
 commit and rollback rule.....432-433  
 commit and rollback scenario.....221  
 commit flow .....214-215, 619  
   on SYNC\_LEVEL(NONE) conversation .....599  
   on SYNC\_LEVEL(SYNCPT) conversation...600  
 commit on return.....725  
 commit unit of work .....212  
 commit unit of work DDM flow .....214  
 commit/rollback processing.....606, 625  
 commitment of work .....212  
 communication connection.....458  
 communication outage notification.....513  
 communications manager .....70, 513, 575, 611  
 CONNECT statement.....46, 54-55  
 connection allocation .....605  
 connection allocation rules.....429, 624  
 connection failure .....431  
 connection usage rule .....436  
 connectivity .....46, 68, 726  
 consistency token .....420  
 continue preceding triplet.....248, 259, 269  
 control-operator verb.....576  
 conversation .....580, 726  
 conversation allocation rule.....429  
 conversation failures.....602  
 conversation flow .....429, 484  
 conversation level.....578  
 conversation protocol error.....559  
 conversation rule .....429, 436  
 conversation verb .....576  
 conversation verb category .....578  
 conversation-level security .....578, 580  
 correlation.....540  
 correlation displays.....543  
 correlation of diagnostic information.....603  
 correlation token.....539, 544, 604, 624  
 correlation, alerts, and supporting data.....604  
 correlation, focal point messages .....544  
 CPT .....248, 259, 269  
 CR rule .....432  
 creating a package.....127  
 crrtkn .....95, 604  
 CU rule.....436  
 cursor disposition.....714  
 cursor error condition .....560  
 cursor position  
   management.....712  
   rules .....713  
 data collection .....604  
 data definition and exchange .....247  
 data descriptor.....253-254  
 data format rule .....444  
 data integrity .....726  
 data representation transformation.....448-450  
 data representation transformation rule ..448, 450  
 data server processing .....715  
 data staging area.....33  
 data stream structure error .....548  
 data stream syntax error.....565  
 data type conversion rule.....440  
 data types.....30  
 Database Management System (DBMS) .....726  
 database partition.....726  
 database server .....70  
 database server (DS) .....726

database support protocol.....	726
database-directed access.....	29
database-directed distributed unit of work.....	726
date.....	352
DBCS.....	726
DBCS (GRAPHIC).....	363-365
DBMS.....	726
DC rule.....	440
DCE.....	54, 97, 100, 458, 607, 625, 726
DCE security.....	515, 607
DDM.....	68-69, 81, 88, 575, 616, 726
DDM bind flow.....	616
DDM command objects in DRDA.....	41
DDM concepts.....	39-40
DDM reader guide.....	39
DDM servers.....	70
deadlocks.....	212
DEALLOCATE.....	543, 548, 577, 596, 604, 608
deallocation type.....	543, 596, 604
decimal floating point.....	375
default triplet.....	274
degrees of distribution.....	48
delete packages.....	141
DESCRIBE.....	196
describe information.....	145, 208
describe input.....	29
describe table.....	200
descriptor classes.....	259
descriptor definitions.....	262
descriptor object.....	248
DF rule.....	444
dgrioprl.....	128
diagnostic data collection.....	604
diagnostic information	
collection and correlation.....	543
diagnostic support.....	84
diagnostics.....	26
dictionary manager.....	73, 81
directory manager.....	73
disconnect.....	465
display.....	603
Distributed Computing Environment (DCE).....	726
Distributed Data Management.....	575
distributed request.....	48, 726
distributed transaction processing.....	21
distributed unit of work.....	35, 46, 620, 727
double-byte character set (DBCS).....	727
downlevel requester.....	715
downstream connection.....	86
DRDA.....	427, 727
DRDA Connection.....	727
DRDA implementation.....	40
DRDA levels.....	2
DRDA managers.....	70
DRDA rules.....	427
DRDA security flows.....	97
DRDA types.....	6
drop package.....	139
DRPPKG.....	75, 139, 256
DS.....	727
DSCERR alert.....	557
DSCINVRM.....	394, 465, 503
DSCPVL.....	256
DSCRDBTBL.....	75, 200-201, 256
DSCSQLSTT.....	75, 198-199, 256
DT rule.....	448
DTAMCHRM.....	300, 304, 394, 465, 503
DTP interface.....	21
DTP Reference Model.....	45
duplicate cursors.....	147
dynamic commit and rollback.....	212
dynamic data format.....	3
Dynamic Data Format.....	727
dynamic execution.....	202
dynamic rollback.....	226, 228, 230
dynamic SQL.....	94, 195, 197, 202, 727
dynamic SQL scenario.....	225-226, 229
early array descriptor.....	277
early data unit descriptors.....	399
early descriptors.....	253, 277
early environmental descriptors.....	333
early group descriptors.....	296
early row descriptors.....	285
eight-byte basic float.....	341
eight-byte integer.....	347
elapsed time.....	542
encrypted UID, password.....	522
encrypted UID, password, new password.....	523
end user.....	607
end-user name.....	416
end-user name rule.....	469
ENDBND.....	75, 127, 131, 256, 590, 616
ENDQRYRM.....	152, 405, 503, 594, 617
ENDUOWRM.....	216, 432, 503, 606, 619
enhanced bind options.....	33
enhanced security.....	33
enhanced sync point manager.....	33
enterprise code.....	417
environmental description.....	251, 253
environmental description objects.....	397
environmental descriptor.....	259
environmental descriptors.....	397
error checking.....	394
error condition to alert model mapping.....	546
error reporting.....	394

- EUN rule.....469
- exact query block.....474
- exchange server attribute .....84, 253, 436  
.....583, 589, 615
- EXCSAT .....84, 97, 253, 277, 436, 583, 589, 615
- EXCSATRD.....84, 97, 277
- EXCSQLIMM.....75, 202-203, 219, 256, 402, 432
- EXCSQLSTT.....75, 171, 195, 215, 253-254  
.....432, 583, 589, 615
- EXCSQLSTT chaining.....16
- execute SQL statement.....171
- EXECUTE\_IMMEDIATE.....402
- execution.....727
- execution flow .....593-594, 617
- execution thread .....84, 727
- EXTDTA.....248
- extended describe.....13
- Extended Privilege Attribute Certificate (EPAC).....27
- Extended Registry Attribute (ERA).....727
- extnam.....92
- failover .....723
- failover support .....28
- failure causes.....550
- failure notification .....575
- FD:OCA .....45, 69, 247, 387, 394, 728
- FDODSC .....248, 261
- FDODTA.....248
- FDOEXT.....248
- FDOOFF.....248
- fixed binary .....377
- fixed byte .....355
- fixed character mixed.....366
- fixed character SBCS .....360
- fixed decimal.....343
- fixed row protocol .....59, 148
- fixed-character .....363
- FIXROWPRC.....59, 156, 485, 589, 616
- flexible blocking .....153-154
- flexible query block.....474
- float.....340-342
- flow .....53, 57-59, 61, 63-64, 89, 127, 202-203, 728
- focal point messages .....539, 545, 624
- four-byte basic float.....342
- four-byte integer.....337
- GDA .....251, 259, 269, 335-336, 393
- GDA/CPT errors.....395
- general errors.....395
- generating alerts.....545
- generic focal point messages .....545
- GENERR alert .....558
- GET\_ATTRIBUTES .....577, 582-583
- GET\_TP\_PROPERTIES.....577, 582-583
- global transactions .....235
- group data array.....251, 259, 269, 335-336, 393
- GSS-API .....515, 728
- gss\_accept\_security\_context.....516
- gss\_accept\_sec\_context.....731
- gss\_init\_security\_context.....516
- gss\_init\_sec\_context.....515-516, 731
- GTRID .....235
- handling conversation failures.....602
- hopping.....86
- host variable .....728
- I/O parallelism .....36
- ID number alert .....550
- immediate SQL statement execution.....202
- initialization flow .....54, 583, 585, 589, 614
- initializing a conversation.....580
- initializing a TCP/IP connection .....614
- input variable array.....25
- INSERT command.....406
- install causes .....550
- instantiate .....728
- integer .....337-339
- intermediate server processing .....530
- interrupt request.....15
- INTRDBRQS.....75, 256
- IPv6.....6
- IR rule.....452
- Kerberos.....26
  - with GSS-API .....525
  - with SSPI.....525
- Kerberos protocol .....525
- Kerberos security mechanism.....525
- large object bytes .....381
- large object bytes locator .....348
- large object character
  - DBCS (GRAPHIC).....383
  - DBCS locator .....350
  - locator.....349
  - SBCS .....382
- large object character mixed .....384
- late array descriptors .....263
- late data unit descriptors.....401
- late descriptor .....253-254, 263
- late environmental descriptor .....335, 385
- late group descriptors.....269
- late row descriptors.....266
- Level 1 .....35
- Level 2 .....35
- Level 3 .....32
- Level 4 .....29
- Level 5.....9-10
- Level 6 .....3
- LID.....251, 259, 269, 334, 393-394, 728

LID example .....	393
LID mapping .....	14
like .....	728
limited block fetch .....	157
limited block protocol .....	59, 164
LMTBLKPRC .....	59, 157, 163, 589, 616
LOB externalization .....	274
LOB processing .....	17
local identifier .....	251, 259, 334, 393-394
local identifier (LID) .....	728
local transactions .....	235
logical flow .....	589, 616
logical unit (LU) .....	728
logical unit of work .....	603
logical unit of work (LUW) .....	728
logical unit of work identifier .....	54, 580
logical unit of work identifier (LUWID) .....	728
Logical Unit type 6.2 (LU 6.2) .....	728
LOG_DATA .....	596-597
long identifiers .....	9
long variable bytes .....	357
long variable character .....	365
long variable character mixed .....	368
long variable character SBCS .....	362
LU .....	728
LU 6.2 .....	596, 602, 728
base and option sets .....	577
initialization flow .....	583
initialization processing .....	580
LU 6.2 (Logical Unit 6.2) .....	543, 575-576
.....	603, 607-608
LU 6.2 flow .....	589-590, 594
LU-LU verification .....	580
LUNAME .....	580
LUW .....	729
LUWID .....	54, 580, 603-604, 608, 729
LUW_Identifier .....	578
LUW_IDENTIFIER .....	582
LU_NAME .....	580
major subfield construction .....	566
major subvector construction .....	566
major vector construction .....	566
Management Services Architecture .....	45
managing conversation .....	602
mapping reply messages .....	546
materialization rules .....	394
MAXBLKEXT .....	177
MAXRSLCNT .....	177
MAXSCTNBR .....	460
MBCS .....	729
MDD .....	248, 259, 333, 385, 391, 394
MDD errors .....	395
message models .....	545
meta data definition .....	248, 259, 333, 385, 391, 394
meta data summary .....	387
MGRDEPRM .....	503, 602
MGRLVLRM .....	503
mixed-byte character set (MBCS) .....	729
mixed-byte datalink .....	374
model mapping .....	546
modification levels .....	8
monitoring .....	9, 541
MSA .....	45, 729
multi-RDB scenario .....	230, 232
multi-relational database update .....	217, 229, 234
multi-row fetch .....	36, 484
multi-row input .....	15
multi-row insert .....	36
Multilingual Latin-1 .....	385
mutual authentication .....	729
n-flow security authentication .....	7
name syntax .....	417
name tables and views .....	418
naming conventions .....	415, 425, 610
network connection .....	24, 54, 217, 221, 618-619, 729
network connectivity flag .....	541
network management .....	539, 603
null-terminated bytes .....	358
null-terminated mixed .....	369
null-terminated SBCS .....	359
numeric character .....	345
OBJDSS .....	81, 83, 479, 615
object data stream structure .....	81, 83, 479, 615
object name row .....	486
object name rule .....	469
object-oriented extensions .....	31
OBJECTID .....	418
OBJNSPRM .....	453, 503
OC rule .....	453
ON rule .....	469
one-byte integer .....	339
operations .....	709
OPNQFLRM .....	503
OPNQRY .....	59, 75, 148, 150, 253-254, 256, 403, 484
OPNQRYRM .....	148, 157, 161, 503, 594, 617
option set functions .....	578
option set of LU 6.2 .....	577
optionality .....	453
optionality rules .....	453
original password substitute .....	520
OUTOVR .....	248
override triplet .....	274, 393
overriding descriptor .....	391
overriding everything .....	391
overriding output formats .....	274

- overriding user data.....**392**  
 package.....**57-58, 127, 419, 729**  
 package collection resolution .....**421**  
 package consistency token.....**420**  
 package management .....**5**  
 PACKAGEID.....**419**  
 packet flow .....**65**  
 partitioned database .....**729**  
 Pascal L string.....**371**  
 Pascal L string bytes.....**370**  
 Pascal L string mixed.....**372**  
 passing USER\_ID.....**416**  
 passing warning to application requester rule.**468**  
 PASSWORD.....**581**  
 PB rule.....**454**  
 ping.....**542**  
 pkgathrul .....**128, 144**  
 PKGBNARM .....**503**  
 PKGBPARM .....**129, 503**  
 plan.....**729**  
 plug-in security .....**11**  
 port .....**729**  
 post-commit processing .....**232**  
 POST\_ON\_RECEIPT .....**581-582, 589**  
 PRCCNV alert.....**559**  
 PRCCNVRM .....**465, 503**  
 PREPARE\_TO\_RECEIVE .....**578, 581, 589**  
 principal.....**729**  
 PRMNSPRM.....**453, 503**  
 probable causes.....**550**  
 problem determination .....**539, 603-604, 623-624**  
     and isolation .....**543**  
 process model .....**70**  
 process model flow .....**81-82**  
 processing model.....**68, 70**  
 product-unique extensions .....**84**  
 program binding rule .....**454**  
 program name.....**425**  
 program preparation process .....**729**  
 program to program communication.....**575**  
 progressive reference .....**729**  
 protected connection.....**23**  
 protected conversation .....**730**  
 protected network connection.....**730**  
 protected resource .....**730**  
 PROTECTED\_LUW\_IDENTIFIER.....**578, 582**  
 protocol .....**59, 730**  
 PRPSQLSTT.....**75, 196, 256, 432**  
 PWDENC.....**30**  
 PWDSBS.....**30**  
 QP rule .....**484**  
 qryblkctl.....**150**  
 qryblksz.....**150-151, 155**  
 QRYDSC .....**248, 254, 484**  
 QRYDTA .....**153, 248, 254**  
 QRYERR alert.....**560**  
 QRYNOPRM .....**161, 465, 503**  
 QRYPOPRM .....**152, 465**  
 QT rule .....**490**  
 query block.....**474**  
 query block chaining rule.....**478**  
 query block format rule.....**474**  
 query block size rule .....**477**  
 query blocks .....**473**  
 query data.....**248**  
 query data transfer protocol rule .....**484**  
 query flow .....**127, 148**  
 query instance.....**145, 482**  
 query instance identifier.....**19**  
 query instance rule.....**482**  
 query option.....**16**  
 query process .....**145**  
 query processing rules.....**18**  
 query protocol.....**710**  
 query terminate, interrupt, continue rule .....**490**  
 query termination rule.....**430**  
 RDB.....**76, 723, 730**  
     accessing .....**89**  
     RDB initiated rollback rule.....**452**  
     RDB initiated rollback scenario .....**233**  
     RDBACCRM .....**465, 503**  
     RDBAFLRM .....**503**  
     RDBATHRM.....**503**  
     RDBCMM.....**75, 214-215, 256**  
     RDBERR alert.....**561**  
     RDBMS.....**45**  
     RDBNACRM.....**465, 503**  
     RDBNFNRM .....**503**  
     RDBRLLBCK.....**75, 256, 600, 620**  
     RDBUPDRM.....**219, 467, 503**  
     RDB\_NAME .....**417, 419, 730**  
     RDB\_NAME rule .....**470**  
     READ socket call .....**617**  
     REBIND .....**75, 143, 256**  
     RECEIVE operations.....**581, 589-590**  
     RECEIVE\_AND\_WAIT .....**577, 583, 593-594, 597**  
 referencing overrides .....**392**  
 referencing rule .....**392**  
 relational data .....**730**  
 relational database access error.....**561**  
 relational database manager.....**76, 82, 91, 93**  
 relational database name rule.....**470**  
 relational database names .....**417**  
 relational database names rules .....**626**  
 relational database-initiated rollback rule .....**452**

- remote unit of work .....48, 619, 730
- replay.....730
- replicated RDB.....723
- reply data object.....74, 82
- reply data stream structure.....83, 405
  - .....486, 589, 594, 615, 617
- reply message.....95, 214, 594
- reply messages.....431
- reply objects and messages .....42
- request correlation identifier .....81, 83
- request packet object.....541
- requester
  - downlevel.....715
- required base set functions .....577
- required option set function .....578
- resource limit reached.....548, 562
- resource recovery.....61, 75, 221, 429, 432, 602, 730
- resource recovery interface .....212, 229
- resource sharing.....23
- response packet object .....541
- result set component.....730
- result set locator.....346
- result table .....707
- resynchronization manager .....73
- returning SQL diagnostics .....204
- RLO .....247-248, 251, 259, 261, 263, 266, 393
- RLO errors .....396
- RN rule.....470
- robust .....730
- ROLLBACK.....212
- rollback.....214
- ROLLBACK .....218-219
- rollback flow.....620
  - on SYNC\_LEVEL(NONE) conversation .....600
  - on SYNC\_LEVEL(SYNCPT) conversation...601
- rollback unit of work.....214
- row identifier.....351
- row layout .....247-248, 251, 259, 261, 263, 266, 393
- rowset .....711
- rowset cursor.....28, 719
- rowset processing.....719
- RPYDSS.....83, 405, 486, 589, 594, 615, 617
- RQSDSS.....81
- RSCLMT alert.....562
- RSCLMTRM .....503
- RSLSETFLG.....177
- RSLSETRM .....177, 503
- RSYNCMGR.....73
- Rule CA5.....429
- rule usage .....605-607
- rule usage for SNA.....605
- rule usage for TCP/IP environments .....624
- rule usage of relational database names .....609
- rules for CLOSE .....500
- rules for CLSQRY .....492
- rules for CNTQRY .....492
- rules for EXCSQLSTT .....492
- rules for FETCH.....497
- rules for OPNQRY.....492
- running DRDA request.....209
- SBCS .....359-362, 371, 469, 505, 730
- SBCS datalink.....373
- scrollability .....707
- scrollable cursor.....12, 36, 707
- SDA .....247, 253, 333, 385, 393
- SE rule .....458
- SECCHK .....97, 585, 615
- SECCHKRM.....97, 465, 503, 585, 615
- SECMGR.....72, 82
- section number assignment rule .....460
- security .....33, 54, 97, 111, 117, 416, 578, 607
- security context information.....515, 731
- security flow .....100
- security manager.....71-72, 82, 91
- security mechanism .....26
  - encrypted UID and password .....522
  - encrypted UID, password, new password ...523
  - user ID and encrypted password.....521
  - user ID and password.....517
  - user ID and password substitute .....520
  - user ID, password, new password.....518
  - user ID-only.....519
- security mechanisms.....10, 30
  - Kerberos.....525
- security rule.....458
- security violation error .....564
- security-sensitive data.....111, 117
- SECURITY\_USER\_ID .....582
- SECVIOL alert.....564
- semantics .....731
- SEND\_DATA .....577, 581, 583, 589-590, 593
- SEND\_ERROR .....577
- sensitivity.....708
- server list.....34
- server processing.....85
- serviceability rules.....465, 608, 625
- SET statement.....464
- SET\_SYNCPT\_OPTIONS .....579
- severity codes.....85, 546
- simple data array.....247, 253, 333, 385, 393
- single relational database update.....217
- single-byte character set.....359-362, 371, 469, 505
- single-byte character set (SBCS).....731
- site processing.....29
- sixteen-byte basic float.....340

SN rule .....	460	SQLERRRM .....	436, 503
SNA .....	45, 54-55, 64, 71, 575, 731	SOLEXTGRP .....	319
SNA environment usage .....	603	SOLEXTROW .....	318
socket .....	731	SQLNUMEXT .....	317
socket calls .....	617	SQLNUMGRP .....	301
sockets interface .....	613	SQLNUMROW .....	289
source requester .....	85	SQLOBJGRP .....	300
SP rule .....	463	SQLOBJNAM .....	200, 256, 288
special register .....	464	SQLRSGRP .....	296
SQL .....	45, 54-55, 59, 96, 129, 202	SQLRSLRD .....	177, 256, 278
.....	212, 420, 469, 593, 617, 731	SQLRSROW .....	285
SQL application manager .....	74, 91	SQLSTATE .....	100, 155, 219, 429, 431, 503
SQL Connection .....	731	SQLSTT .....	127, 129, 196, 202, 256, 287, 392, 402
SQL EXECUTE IMMEDIATE .....	203	SQLSTTGRP .....	299
SQL EXECUTE_IMMEDIATE .....	432	SQLSTTVRB .....	127, 256, 280, 392
SQL identifier .....	14	SQLTOKGRP .....	332
SQL long identifiers .....	9	SQLTOKROW .....	295
SQL object name rule .....	469	SQLUDTGRP .....	308
SQL object names (ON rules) .....	469	SQLVRBGRP .....	297
SQL statement .....	13	SQLVRBROW .....	286
SQL statement execution flow .....	593, 617	SRRBACK .....	234
SQLAM .....	74, 277	SRRCMIT .....	229
SQLAM (SQL application manager) .....	76, 80, 83	srvclsnm (server class name) .....	91
SQLCA .....	85, 402, 405, 431	srvdgn .....	85
SQLCADTA .....	268	SSCP .....	731
SQLCAGRP .....	302	ST rule .....	464
SQLCARD .....	26, 127, 131, 171, 202, 215	staging area .....	33
.....	256, 290, 405, 486	standard focal point messages .....	623
SQLCAXGRP .....	303	standardized object name .....	424
SQLCINRD .....	256, 279	statement attributes .....	27
SQLCNGRP .....	324	statement execution flow .....	593, 617
SQLCNROW .....	293	statement execution logical flow .....	59
SQLDA .....	334	static commit .....	231
SQLDAGRP .....	306	static commit and rollback .....	212
SQLDARD .....	196, 198, 200, 256, 281	static rollback .....	228, 232
SQLDAROW .....	291	stored procedure .....	34
SQLDCGRP .....	326	stored procedure DDM flow .....	216
SQLDCROW .....	294	stored procedure name .....	423
SQLDCTOKS .....	282	stored procedure result set	
SQLDCXGRP .....	330	scrolling .....	714
SQLDHGRP .....	310	stored procedures rule .....	463
SQLDHROW .....	292	streaming .....	18
SQLDIAGCI .....	283	strong password substitute .....	520
SQLDIAGCN .....	284	Structured Query Language .....	45
SQLDIAGGRP .....	305	sttstrdel .....	94
SQLDIAGSTT .....	320	subfield .....	552
SQLDOPTGRP .....	313	subvector .....	552
SQLDTA .....	148, 157, 171, 249, 253, 267	summary component .....	731
SQLDTAGRP .....	267, 271	supervisor .....	72, 78
SQLDTAMRW .....	256, 265	SV rule .....	465
SQLDTARD .....	171, 174, 177, 256, 264	svrcod .....	85
SQLDXGRP .....	315	switch ID .....	123
		sync point communications manager .....	71

sync point manager	33, 48, 73, 232, 596
synchronization point	61, 576
synchronization point (sync point)	731
synchronization point manager	731
synchronous wait protocol verbs	583
SYNCMNT	596
SYNCPTMGR	48, 73, 212, 234, 596-597, 602
SYNCPTOV	33
SYNC_LEVEL	580, 596, 602, 605
SYNERRCD	436
synonym	424
SYNTAX alert	565
SYNTAXRM	436, 465, 503
system services control point (SSCP)	731
Systems Network Architecture	575
Systems Network Architecture (SNA)	731
TAKE_BACKOUT	234
TAKE_SYNCPT	220
target program	425
target program name	471
target program name (TPN)	731
target program name rule	471
target server	85
TCP/IP	611-612, 732
TCP/IP and DRDA	614
TCP/IP communications	34
TCP/IP communications manager	71, 611
TCP/IP connection	614, 626
TCP/IP connection rule usage	625
TCP/IP correlation value display	623-624
TCP/IP environment usage in DRDA	623
TCP/IP flow	617, 619
TCP/IP initialization flow	615
TCP/IP initialization processing	614
TCP/IP packet flow	616
TCPCMNI	614
TCPCMNT	618
TCPSRCCD	616
TCPSRCCR	616
terminate, interrupt, continue rule	490
terminating conversations	596-597
terminating network connection	618-619
termination	64
termination flows	63
time	353
timely failure notification	575
timestamp	354
token	604
tool and program	603
TP	732
TPN	54, 70-71, 471, 732
TPN rule	471
transaction	732
transaction component	732
transaction pooling	240
transaction processing	45, 514
transaction program (TP)	732
transaction program name	54, 70-71, 609-610, 732
transparency	80
transport control protocol	612
transport control protocol/internet protocol	611
TRGNSPRM	503
triplet	248, 251, 259, 269, 333, 385, 391-394, 732
triplet override	336
trusted application server	7, 119, 527
trusted connection	119
two-byte integer	338
two-phase commit	61, 73, 213, 596, 602
two-phase commit protocols	732
TYPDEFNAM	127, 143, 277, 334, 391, 429, 448
typdefovr	95
TYPDEFOVR	127, 171, 277, 333, 429, 449
type-independent verb	576-577
unique identifier	603
unit of work	67, 214, 430, 466, 603, 732
Universal Unique Identifier (UUID)	732
unlike	733
unprotected conversation	733
unprotected downstream updates	7
unprotected network connection	733
UOWID (unit of work identifier)	54
UP rule	467
updatability	709
update control rule	467
update privilege	213, 217
upstream connection	86
usage of names	609
user causes	550
user ID and encrypted password	521
user ID and password	97, 100, 517
user ID and password substitute	520
user ID only	97, 100
user ID security	516
user ID verification	578
user ID, password, new password	518
user ID-only	519
USER_ID	416, 581
utility flows	65
UUID	733
VALNSPRM	151, 430, 450, 453, 503
variable array	406
variable binary	378
variable byte	356
variable character	364
variable character mixed	367

variable character SBCS.....	361
verb.....	610
verb categories.....	576
verb, LU 6.2.....	576
version ID.....	421
version management.....	420
well known port.....	71, 733
WN rule.....	468
WRITE socket call.....	617
XA.....	21
XA manager.....	77
XML extensions.....	3
XML string external encoding.....	380
XML string internal encoding.....	379
zoned decimal.....	344