

***/ Technical Standard***

**DRDA, Version 3, Volume 3:**

**Distributed Data Management (DDM) Architecture**

*The Open Group*



© January 2004, 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 3, Volume 3: Distributed Data Management (DDM) Architecture

ISBN: 1-931624-42-9

Document Number: C045

Published in the U.K. by The Open Group, January 2004.

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

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

or by Electronic Mail to:

OGSpecs@opengroup.org

# Contents

<b>Chapter 1</b>	<b>The DRDA Specification .....</b>	<b>1</b>
1.1	DRDA Reference .....	2
1.2	The FD:OCA Reference .....	2
1.3	The DDM Reference .....	3
<b>Chapter 2</b>	<b>Introduction to DDM .....</b>	<b>7</b>
2.1	DDM Architecture .....	7
2.2	DDM is an Open Architecture .....	8
2.3	Benefits of Using DDM .....	8
2.3.1	Reduced Programming Costs .....	8
2.3.2	Reduced Data Redundancy .....	8
2.3.3	More Timely Information.....	9
2.3.4	Local/Remote Transparency.....	9
2.3.5	Better Resource Management .....	9
2.3.6	Data Integrity.....	9
2.3.7	Standardization .....	10
2.4	The Basic Structure of DDM.....	10
2.4.1	The Language of DDM .....	10
2.4.2	DDM Components .....	10
2.4.3	How DDM Works.....	11
2.4.4	The DDM Relational Database Model.....	11
2.4.4.1	Structure of the Distributed Processing Model .....	12
2.4.4.2	RDB Characteristics.....	12
2.4.4.3	SQLAM Characteristics .....	13
2.5	DDM Programming Techniques.....	13
2.5.1	Data.....	14
2.5.2	Object.....	14
2.5.3	Server.....	14
2.5.4	Manager .....	14
2.5.4.1	Communications Manager .....	15
2.5.4.2	Agents .....	15
2.5.4.3	Dictionary.....	16
2.5.4.4	Security Manager .....	16
2.5.4.5	Supervisor .....	16
2.5.4.6	RDB Manager.....	16
2.5.4.7	SQL Application Manager .....	17
2.5.4.8	Data Conversion .....	17
2.5.4.9	Sync Point Manager.....	17
2.5.4.10	XA Manager .....	18
2.5.5	Connectivity.....	18
2.5.5.1	Data Connectivity.....	19
2.5.5.2	Negotiation .....	19

2.5.5.3	Selecting Subsets.....	20
2.5.5.4	Developing Product-Unique Extensions .....	22
2.5.6	DDM Object-Oriented Programming.....	22
2.5.6.1	Object.....	23
2.5.6.2	Class.....	23
2.5.6.3	Class Inheritance.....	24
2.5.6.4	Grammar.....	24
2.5.6.5	Protocols .....	24
2.5.7	DDM Dictionaries.....	25
2.5.7.1	Dictionary Contents .....	26
2.5.7.2	Tutorial Objects Dictionary.....	26
2.5.7.3	Primitive Classes Dictionary .....	27
2.5.7.4	Base Classes Dictionary.....	27
2.5.7.5	Relational Database Classes Dictionary.....	27
2.5.8	Using the DDM Reference Manual .....	28
<b>Chapter 3</b>	<b>Terms .....</b>	<b>29</b>
<b>Appendix A</b>	<b>Codepoints (Sorted by Term Name).....</b>	<b>1119</b>
<b>Appendix B</b>	<b>Term Names (Sorted by Codepoints).....</b>	<b>1133</b>
<b>Appendix C</b>	<b>DDM Codepoint Scheme.....</b>	<b>1147</b>
	<b>Index.....</b>	<b>1149</b>
 <b>List of Figures</b>		
3-1	Communications Failure Between Source and Target.....	69
3-2	Communications Initiation by Source System .....	73
3-3	Normal Communications Termination.....	76
3-4	Source Sends Command with Data (APPSRCCD) Protocol (Part 1) ..	80
3-5	Source Sends Command with Data (APPSRCCD) Protocol (Part 2) ..	81
3-6	Source Sends Command with Data (APPSRCCD) Protocol (Part 3) ..	82
3-7	Source Sends Command with No Data (APPSRCCR) Protocol (Part 1) .....	87
3-8	Source Sends Command with No Data (APPSRCCR) Protocol (Part 2) .....	88
3-9	Source System Detects Error (APPSRCER) Protocol.....	92
3-10	Target System Detects Error (APPTRGER) Protocol (Part 1) .....	96
3-11	Target System Detects Error (APPTRGER) Protocol (Part 2) .....	97
3-12	Superclass, Class, and Instance Relationships.....	159
3-13	Class to Metaclass Relationships.....	160
3-14	Superclass and Super-Metaclass Chains.....	161
3-15	Class Structure Overview .....	162
3-16	DDM as a Layered Communications Architecture .....	195
3-17	Server Data Paths for Access Method Services .....	196
3-18	Codepoint References to Dictionaries .....	236

3-19	Collection-to-Data-Stream Mapping .....	238
3-20	DCE-Based Security Flows Using GSS-API .....	253
3-21	DDM DCE Security Flows .....	254
3-22	Structure of a Definition List .....	268
3-23	Dictionary Structure Overview .....	286
3-24	Layered Communications Reference Model.....	308
3-25	Layers (A, B, and C) in the DDM Architecture .....	309
3-26	DDM Layer A Data Stream Structure.....	309
3-27	RQSDSS Chaining Rules .....	311
3-28	RPYDSS and OBJDSS Chaining Rules.....	312
3-29	RQSDSS Chain Error Termination .....	312
3-30	Mapping Small DDM Layer B Objects to Layer A DSSs .....	313
3-31	Mapping Large DDM Layer B Objects to Layer A DSSs .....	314
3-32	Summary of DDM Data Stream Structure Layers .....	316
3-33	DSS Format Byte Layout .....	318
3-34	DDM Encrypted User ID Password Data Security Flows.....	323
3-35	DDM Encrypted User ID, Password, and Security-Sensitive Data.....	325
3-36	Data Encryption Flow for Intermediate Server .....	326
3-37	Intermediate Server when Upstream Site does not Support Encryption.....	327
3-38	An Example of Package Creation by ENDBND .....	337
3-39	FD:OCA Processing Overview .....	404
3-40	Inheritance in a Zoological Taxonomy.....	437
3-41	Inheritance in the DDM Taxonomy .....	438
3-42	Levels of the DDM Descriptive Hierarchy.....	439
3-43	Encoded DATA Class Hierarchy .....	439
3-44	Data Object Class Hierarchy .....	440
3-45	Object Management Classes Hierarchy.....	440
3-46	Manager Control Hierarchy .....	441
3-47	Example of Kerberos-Based Flow using SSPI and GSS-API.....	466
3-48	DDM Kerberos Security Flows .....	467
3-49	Mapping a DDM Server onto a System.....	515
3-50	Source Server Manager Interactions.....	515
3-51	Target Server Manager Interactions.....	516
3-52	Data Objects .....	536
3-53	A Single Record with Record Number Feedback .....	537
3-54	A Single Record Formatted in a Continued OBJDSS.....	537
3-55	DDM Objects.....	545
3-56	DDM Object Interchange Format .....	546
3-57	Objects—Data Items Encapsulated by Programs .....	547
3-58	Objects as Instances of a Class .....	548
3-59	Hierarchical Taxonomy .....	549
3-60	A Sample of the DDM Class Hierarchy .....	550
3-61	Inheritance of Programs from More Abstract Classes .....	551
3-62	Smalltalk Instance, Class, and Metaclass Relationships.....	552
3-63	DDM Instance, Class, and Metaclass Relationships.....	553
3-64	Example of Plug-In-Based Flows.....	615

3-65	DDM Plug-In Security Flows .....	616
3-66	Transparent Data Management Services .....	628
3-67	DDM Processing Overview .....	629
3-68	Application to Local RDB Connection .....	741
3-69	Application to Remote RDB Connection.....	742
3-70	Reply Data Stream Structures .....	766
3-71	Data Stream Structure Request Correlation.....	773
3-72	Request Data Stream Structures .....	774
3-73	Server Paths for RSYNCMGR at DDM Level 5 .....	788
3-74	Scalar to Data Stream Mapping.....	796
3-75	Processing Model for Programs with Embedded SQL Statements .....	840
3-76	Application Requester and a Remote RDB .....	847
3-77	Application Requester, Remote RDBs, and Two-Phase Commit .....	848
3-78	DDM Distributed Systems and Servers .....	881
3-79	Structural Layers in Nature .....	891
3-80	The Structural Layers of DDM Architecture.....	892
3-81	Backout Flows Between Managers (SYNCMNBK) Protocol .....	924
3-82	Two-Phase Commit Flows Between Managers (SYNCMNCM) Protocol.....	926
3-83	Communications Failure Between Source and Target (SYNCMNFL) .....	929
3-84	Communications Initiation by Source System (SYNCMNI) .....	932
3-85	Normal Communications Termination (SYNCMNT) Protocol.....	936
3-86	Server Paths for SYNCPTMGR at DDM Level 4.....	940
3-87	Server Paths for SYNCPTMGR at DDM Level 5.....	941
3-88	Illustration of Sync Point Flow.....	946
3-89	Illustration of Resync Server Sync Point Flow.....	947
3-90	Multiple Released Connections Commit Flow.....	951
3-91	Multiple Released Connections Rollback Flow.....	952
3-92	Commit With Updates Flow .....	953
3-93	Commit Without Updates Flow .....	953
3-94	Released Connection Commit With/Without Updates .....	954
3-95	Source SYNCPTMGR Initiated Rollback.....	954
3-96	Target SYNCPTMGR Initiated Rollback Flow .....	955
3-97	Source SYNCPTMGR Terminates a Conversation.....	955
3-98	Target SYNCPTMGR Initiated Rollback Flow .....	956
3-99	Commit Using an Implied Forget Flow .....	957
3-100	Commit Flows SYNCPTMGR Without Log .....	958
3-101	Rollback for SYNCPTMGR Without Log .....	959
3-102	Multiple Released Connections Commit Flow.....	960
3-103	Multiple Released Connections Rollback Flow.....	961
3-104	Set of Connections that can Share Resources .....	962
3-105	Resync Connection (Part 1).....	966
3-106	Resync Connection (Part 2).....	967
3-107	Multiple Connections Race Condition (Part 1).....	969

3-108	Multiple Connections Race Condition (Part 2).....	970
3-109	Sub-Cell Contents.....	971
3-110	Communications Failure Between Source and Target.....	992
3-111	TCP/IP Communications Initiation by Source System.....	995
3-112	Normal Communications Termination.....	997
3-113	TCP/IP Internet Components.....	1000
3-114	Class C Internetwork Address.....	1002
3-115	Source Sends Command With Data .....	1008
3-116	Source Sends Command With No Command Data.....	1011
3-117	Source System Detects Error .....	1013
3-118	Target System Detects Error .....	1016
3-119	The Usage of TYPDEFOVR .....	1031
3-120	DDM USRID Security With a Password.....	1050
3-121	DDM USRID Security with a New Password.....	1052
3-122	Requesting XAMGR Support.....	1071
3-123	SYNCCTL(New UOW) Functionality Flow.....	1072
3-124	Global Transactions (Tightly-Coupled) .....	1074
3-125	Global Transactions (Loosely-Coupled) .....	1075
3-126	Resuming Suspended Transaction Branches .....	1076
3-127	Example of TMJOIN Usage .....	1078
3-128	Ending a Global Transaction .....	1079
3-129	Application Server Cannot Suspend Resources.....	1081
3-130	Application Server Can Suspend Resources.....	1082
3-131	Suspend and Migrate .....	1084
3-132	Preparing a Global Transaction.....	1085
3-133	Example of a Prepare from any Connection .....	1087
3-134	Committing a Transaction .....	1088
3-135	Rolling Back a Transaction.....	1091
3-136	Recovery Process .....	1093
3-137	Obtaining a List of XIDs from the Application Server.....	1095
3-138	Forgetting Heuristically Completed Transaction Branches.....	1096
3-139	Example of a Local Transaction .....	1097
3-140	xa_open Requesting XAMGR Support .....	1098
3-141	Sample of Static and Dynamic Registration using SYNCCTL(New UOW) .....	1099
3-142	Sample Flow of xa_end using SYNCCTL(End).....	1100
3-143	Sample Flow of xa_prepare using SYNCCTL(Prepare) .....	1101
3-144	Sample flow of xa_commit using SYNCCTL(Commit) .....	1102
3-145	Sample Flow of xa_rollback using SYNCCTL(Rollback).....	1103
3-146	Sample Flow of xa_recover using SYNCCTL(Return Indoubt List) .....	1104
3-147	Sample Flow of xa_forget using SYNCCTL(Forget) .....	1105
3-148	xa_close.....	1105
3-149	Asynchronous XA Operation.....	1106

**List of Tables**

1-1	DDM Modeling and Description Terms.....	3
-----	---	---

1-2	DDM Terms of Interest to DRDA Implementers .....	4
1-3	DDM Command Objects Used by DRDA .....	5
1-4	DDM Reply Data Objects Used by DRDA .....	6
3-1	Agent-Level Compatibility .....	62
3-2	Required CCSID Support for the CCSIDMGR .....	150
3-3	Required CCSIDs .....	155
3-4	CLSQRX Summary Decision Table .....	167
3-5	Communications Manager (CMNAPPC) SNA LU 6.2 Verbs .....	186
3-6	Communications Manager (CMNSYNCP) SNA LU 6.2 Verbs .....	204
3-7	SNA LU 6.2 Sync Point Conversational Communications .....	212
3-8	TCP/IP Communications Manager (CMNTCPIP) Socket Calls .....	214
3-9	Well-Known Port and Symbolic Name Assignment .....	218
3-10	Sync Point Manager-Level Compatibility .....	788
3-11	SECCHK Valid Security Mechanism Combinations .....	801
3-12	Relationship of SECCHKCD and SVRCOD in the SECCHKRM .....	806
3-13	Valid Security Mechanism Component Combinations .....	811
3-14	Security Manager-Level Compatibility .....	815
3-15	SQL Application Manager-Level Compatibility .....	850
3-16	Supervisor-Level Compatibility .....	908
3-17	Sync Point Manager-Level Compatibility .....	941
3-18	Coordinator SYNCPTMGR With Log .....	972
3-19	Coordinator SYNCPTMGR With No Log .....	976
3-20	Resync Server SYNCPTMGR .....	978
3-21	Internet Address Structure .....	1002



# Preface

## **The Open Group**

The Open Group, a vendor and technology-neutral consortium, has a vision of Boundaryless Information Flow achieved through global interoperability in a secure, reliable, and timely manner. The Open Group's mission is to drive the creation of Boundaryless Information Flow by:

- Working with customers to capture, understand, and address current and emerging requirements, establish policies, and share best practices
- Working with suppliers, consortia, and standards bodies to develop consensus and facilitate interoperability, to evolve and integrate open specifications and open source technologies
- Offering a comprehensive set of services to enhance the operational efficiency of consortia
- Developing and operating the industry's premier certification service and encouraging procurement of certified products

The Open Group provides opportunities to exchange information and shape the future of IT. The Open Group members include some of the largest and most influential organizations in the world. The flexible structure of The Open Group membership allows for almost any organization, no matter what their size, to join and have a voice in shaping the future of the IT world.

More information 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/testing](http://www.opengroup.org/testing).

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/pubs](http://www.opengroup.org/pubs).

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 Data Management Architecture*, 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 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.

## 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 reference of DDM commands, parameters, data objects, and messages as a basis for their interface 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 is a trademark and UNIX and The Open Group are registered trademarks of The Open Group in the United States and other countries.

All other trademarks are the property of their respective owners.

HP-UX<sup>®</sup> is a registered trademark of Hewlett-Packard Company.

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

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

Intel<sup>®</sup> is a registered trademark of Intel Corporation.

Microsoft<sup>®</sup> and Windows NT<sup>®</sup> are registered trademarks of Microsoft Corporation.

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

Solaris<sup>®</sup> is a registered trademark of Sun Microsystems, Inc.

VAX<sup>®</sup> 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 3, Volume 1: Distributed Relational Database Architecture (DRDA), published by The Open Group.

## **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 3, Volume 3: Distributed Data Management (DDM) Architecture, published by The Open Group (this document).
- *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).
- *UNIX Networking*, Kochan and Wood, Hayden Books, Indiana, 1989.

## Referenced Documents

- *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 <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 3, 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*.
- 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.

# 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 among 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 Chapter 2 (on page 7), which describes the overall structure and basic concepts, is read either before reading the chapter in the DRDA Reference entitled “The DRDA Processing Model and Command Flows” or in conjunction with it. The rest of the DDM Reference should be used primarily as a reference when additional detail is needed to implement the functions and flows as defined in the DRDA Reference. Similarly, one can use the overview of FD:OCA below and the introductory section of its respective volume and only refer to the details of the FD:OCA constructs as needed during implementation.

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

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

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

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

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

## **1.1 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.

## **1.2 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 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.

#### 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

<b>DDM Term</b>	<b>Term Title</b>
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 3), the reader needs to understand that every DRDA implementation needs to provide the DDM components and model structures listed in Table 1-2. The concept of a component is described in the overview term for that component.

**Table 1-2** DDM Terms of Interest to DRDA Implementers

<b>DDM Term</b>	<b>Term Title</b>
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
XAMGROV	XA Manager Overview

**DDM Command Objects in DRDA**

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

**Table 1-3** DDM Command Objects Used by DRDA

<b>DDM Term</b>	<b>Term Title</b>
<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
CLSQRY	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 and XA connections
SYNCRSY	Sync point resynchronization request used by DUOW connections
<i>Security processing</i>	
ACCSEC	Access security
SECCHK	Security check

**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

<b>DDM Term</b>	<b>Term Title</b>
ACCRDBRM	Access to RDB completed
ACCSECRD	Access security reply data
BGNBNDRM	Begin bind error
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 information reply data
SQLDTARD	SQL data reply data
SQLRSLRD	SQL result set reply data
SYNCCRD	Sync point control reply data in support of DUOW and XA
SYNCLOG	Identifies the sync point log used for a unit of work
SYNCRRD	Sync point resynchronization reply data in support of distributed unit of work
TYPDEFNAM	Data type definition name
TYPDEFOVR	Data type definition override

# Introduction to DDM

DRDA uses distributed data management as a methodology that allows data stored in a relational database system to be accessed by another system. This requires a consistent set of protocols among different database vendors. Because database vendors store and represent data differently, they are unable to share data in heterogeneous database environments. Without DDM, each implementation must be written for each different relational database.

What these systems need is a methodology they can all use. And that is exactly what DDM architecture provides.

## 2.1 DDM Architecture

DDM architecture makes the sharing and accessing of data between computer systems possible by providing a common language and a set of rules that enable different systems to communicate and share data. The DDM architecture can be used to build cross-system data management capability into new or existing systems. The objectives of DDM are:

- To provide data interchange among different kinds of systems
- To increase efficient data exchange among similar systems
- To standardize data management facilities for new systems

The facets of DDM architecture that allow the reader to reach these objectives include:

- A data connectivity language.

Data connectivity is the ability of systems to exchange data efficiently. DDM has a vocabulary of terms and a set of rules (a grammar) for accessing data from relational databases.

- A standardized relational database model and a Structured Query Language Application Manager (SQLAM).

A relational database model is a description of how data within a relational database is organized and managed.

An SQL Application Manager is a description of a consistent way of requesting SQL services from a relational database.

These features of DDM allow the application program or its user to obtain data without concern for where the file or relational database is located.

## 2.2 DDM is an Open Architecture

System planners can tailor DDM architecture to suit the needs of a particular system. For example, if a system does not support all the file models and access methods that DDM defines, subsets of the DDM architecture can be selected and implemented. Additional DDM features can be added later as needs and system capabilities change.

Extensions can also be added to DDM to meet the unique requirements of a system. For example, when a system is sharing data with another system of the same type, all non-DDM functions of the system can be supported by adding extensions to the DDM architecture; the system is not restricted to the standard features the DDM architecture provides.

## 2.3 Benefits of Using DDM

The following section describes some of the benefits from using DDM:

- Reduced Programming Costs
- Reduced Data Redundancy
- More Timely Information
- Local/Remote Transparency
- Better Resource Management
- Data Integrity
- Standardization

### 2.3.1 Reduced Programming Costs

DDM reduces the amount of programming needed to allow applications on one system to access data on remote systems.

For example, suppose there are four systems on a network, and an inventory application on one system needs to access data from files or relational databases located on the other three systems.

Without DDM, to accommodate the remote access needs, three pairs of programs would have to be written to allow each of the systems to communicate and exchange data.

On the other hand, if DDM was implemented at each system, the three pairs of programs would not need to be written. DDM would handle the communications and interfacing required for the systems to exchange data.

### 2.3.2 Reduced Data Redundancy

With DDM data can be stored in one location and the data then shared with applications on other systems. Storage space is saved since the same data is not duplicated in every system.

For example, a large hospital may have duplicate information about patients in the laboratory and out patient clinic systems. DDM allows these systems to communicate with each other and share the same data, which is stored in only one location.

### 2.3.3 More Timely Information

DDM enables systems to share data, which means users on one system can always have access to the most current information on another system.

For example, suppose that retail stores in three cities maintained their own inventory information locally, and then at the beginning of each week, they each sent their records to a fourth, central system.

The central system would only have up-to-date information at the beginning of each week; by the middle of the week, the information would be out-of-date.

With DDM, the central system can have direct access to data at the other systems, meaning it will always have access to the most current information.

### 2.3.4 Local/Remote Transparency

With some DDM products, it makes no difference to an application whether a file or relational database is stored locally or at a remote system.

The application passes all relational database (RDB) requests for data to a local data management interface (LDMI). If the RDB is stored locally, the LDMI handles the functions. If the RDB is stored at a remote system, the LDMI passes the request to DDM, which handles the communications and performs the RDB requests necessary to retrieve the data.

### 2.3.5 Better Resource Management

Sharing data among systems using DDM can also result in better management of each system's resources.

An example of this is a data processing department with a workload that has grown too large for its direct access storage devices (DASDs) but has not yet reached a point that justifies investing in new equipment. However, another department is not using its full DASD capacity. DDM would allow the manager to transfer some of the data to the second department. The end result is the maximum use of the resources on hand.

### 2.3.6 Data Integrity

DDM also helps to ensure that data will not be accidentally destroyed or altered, and that updates will not be lost because of conflicts between concurrent users.

The DDM architecture includes two-phase commit, which ensures that any changes made to data in an RDB (which is based on data in another RDB) by a user will not be lost.

Both one phase and two-phase commit processing, respectively used for single and multiple site updating, are documented in the DDM architecture. Transactions against the relational database are gathered in Units of Work (UOWID or XID), and the resource recovery process runs at the UOW level. All the activity within a UOW is committed (or backed out) and then a new UOW is started for SYNCPTMGR protected connections. For XAMGR protected connections, a new UOW can be started after a SYNCCTL(End) has been issued.

### 2.3.7 Standardization

Standardization is needed to ensure connectivity. Without standardization, it is difficult for one system to access data from another system.

Items that require standardization include:

- Structured Query Language (SQL) managers
- Communications protocols (TCP/IP or SNA)
- The syntax of requests, replies, and data

DDM provides standardization for the above items, which helps to ensure connectivity among products and systems.

## 2.4 The Basic Structure of DDM

This section takes a look inside DDM. It describes:

- The Language of DDM
- DDM Components
- How DDM Works
- The DDM Relational Database Model

### 2.4.1 The Language of DDM

DDM architecture can be considered a language used for exchanging data between two or more computer systems. Like all languages, DDM contains an ordered system of symbols and rules for their use.

The language of DDM is composed of these three parts:

Vocabulary	A vocabulary is composed of words. In DDM, words are defined terms, such as class and object.
Grammar	In DDM, words are combined in a certain order and obey specific rules. Commands and reply messages are examples of DDM grammatical structures.
Protocol	The DDM protocol is a set of rules used to ensure an orderly exchange of data between two communicating systems.

### 2.4.2 DDM Components

DDM is an architecture, not a software product. However, to use DDM, DDM software products can either be bought or developed. In either case, there will be two types of software components:

- DDM data management services—Software that translates an application's file or relational database requests into DDM commands, and that translates DDM reply messages back into data the application can use.
- DDM communications manager—Software that handles communications procedures necessary to exchange data with another system.



### 2.4.3 How DDM Works

Before describing how DDM processes a request for remote data, the reader will need to know these two terms:

**Source**        The system containing the application program that requests access to data in another system.

**Target**        The system containing the data to be accessed.

The source system has an application program that needs to access files or a relational database located on another system. DDM gives the source system the capability to access data on another system.

The following list traces the processing steps required for the source and target systems to communicate:

1. The application program requests data from a relational database. It does this by sending a request to the local data management interface (LDMI), which first checks whether the requested relational database is stored locally.
2. If the relational database is not found on the local system, the LDMI uses a "trap" to give the data request to the source DDM server, which translates the request into DDM commands.
3. The DDM communications manager on the source system transmits the commands to the DDM communications manager on the target system.
4. The DDM target server does the following:
  - Interprets the DDM commands
  - Locates the requested file or relational database
  - Translates the DDM commands for the local data management interface (LDMI) of the target system
  - Requests execution of the appropriate functions
5. A local data manager (LDM) on the target system retrieves and sends back data through the target LDMI to the target DDM server, which translates the data into DDM form. The DDM communications manager on the target system then transmits the data to the DDM communications manager on the source system.
6. The DDM source server translates the data into the format required by the source system's LDM. The LDM passes the data to the application program.

### 2.4.4 The DDM Relational Database Model

A relational database (RDB) is a database that consists of a collection of tables (relations). DDM architecture defines one relational database model. The structure of the model depends on whether the RDB is located on the local system or on a remote system.

In the single system RDB model, the application that requires Structured Query Language (SQL) services from an RDB sends requests to an SQL Application Manager (SQLAM). The SQLAM provides a consistent method for requesting SQL services from an RDB for a single application.

The SQLAM establishes connectivity with the RDB and accesses the RDB, using other system services such as directories and the security manager (SECMGR). The requested RDB operations are then carried out by the RDB manager. The RDB manager returns the requested data to the SQLAM, which then presents it to the application.

#### 2.4.4.1 Structure of the Distributed Processing Model

In the distributed processing RDB model, the SQLAM, agent, and communications manager services are functionally split between the source and target systems. The source and target systems can be of different types.

The application that requires SQL services from the RDB submits SQL requests to the source SQLAM. The source SQLAM uses source directory services to determine the network location of the RDB to be accessed, then routes the request to the source agent.

The source SQLAM receives any reply coming back from the RDB in response to the SQL request, and presents it to the application. If necessary, the source SQLAM converts the reply to the data representation of the application.

The source agent interfaces with the source communications manager to send SQLAM requests to the target system and to receive replies from the target system. The replies are then passed to the source SQLAM.

The target agent interfaces with the target communications manager to receive requests from the source agent. It validates the requests and routes them to the target SQLAM. The target agent also passes the replies from the target SQLAM back to the source agent.

The target SQLAM manages accesses to the RDB, and performs any necessary data conversions. The internal functions of the RDB manager are not defined in DDM.

#### 2.4.4.2 RDB Characteristics

A relational database (RDB) has the following characteristics:

- The RDB is a manager that manages relational data. It is through this manager that an SQLAM accesses relational data.
- Locking of RDB objects is handled by the RDB manager and is not specified by DDM.
- Among other things, an RDB may contain the following objects:
  - Catalogs
  - Tables
  - Indexes
  - Views
  - Locks
  - Recovery logs
  - Packages
  - Authorizations
  - Cursors
  - RDB collections

### 2.4.4.3 SQLAM Characteristics

Structured Query Language (SQL) is the language used by application programs to access and modify data in a relational database. SQL statements can be:

**Embedded** Contained in the source files of the application programs.

**Dynamic** Typed in from a terminal or built by a program.

The SQL Application Manager (SQLAM) receives SQL statements from an application and performs these services:

- **RDB access.** The source SQLAM uses directory services to locate the correct RDB location on the network, making it unnecessary for the application to know the RDB's location. The target SQLAM accesses the RDB.

In SQLAM level 3, an application can access only one RDB per conversation. To end RDB access, the application must deallocate the conversation.

In SQLAM level 4, an application can access one or more RDBs per conversation. If the systems support the two-phase commit process, multiple RDBs can be updated within one logical unit of work; otherwise, only one RDB can be updated and the remaining RDBs are restricted to read-only access within one logical unit of work.

In SQLAM level 6, an application can obtain a result set when accessing a stored procedure on a remote database.

- **Data conversion.** Different systems represent data differently.

A target SQLAM converts the application's SQL statements to suit the data type for the RDB. The source SQLAM converts data returned by the RDB to suit the application.

- **Query processing.** The source and target SQLAMs work together to efficiently handle SQL statements that can return large amounts of data. Multiple queries can be in process at the same time.

## 2.5 DDM Programming Techniques

The following sections describe terms and concepts used in DDM.

A DDM server is composed of data processing entities called objects. Each object defines its own data structures, what commands it accepts, and how it responds to those commands. Objects that accept the same commands and reply with the same messages are grouped into classes.

Classes are organized in a hierarchy with the most primitive class (DATA) at the highest point in the hierarchy. Classes inherit structure and function from superclasses which are above them in the hierarchy. In this way, classes located further down in the hierarchy become increasingly specialized.

A superclass is a class from which variables and commands are inherited by a subclass. There are four superclasses in the basic hierarchy of DDM architecture. The class DATA is the highest class; all other classes inherit characteristics from the class DATA.

Subclasses inherit the characteristics of the superclasses above them, and also add commands and variables of their own. Because of this, subclasses are always more specialized than their superclasses.

### 2.5.1 Data

The class DATA is the superclass for all of the classes that encode information for DDM. DATA is the most primitive (least specialized) of the DDM classes. Each subclass of the class DATA encodes information according to the definition of the specific subclass. Examples of subclasses of DATA are BITDR (bit data representation), CHRSTRDR (character string data representation), and BINDR (binary number data representation).

### 2.5.2 Object

The class OBJECT is the superclass for all self-identifying entities that structure and store data. Objects that have a common structure and that respond to the same commands are grouped together in the same class. Examples of subclasses of OBJECT are all of the DDM commands (class COMMAND) and reply messages (class RPYMSG).

### 2.5.3 Server

The class SERVER allocates and controls the space, existence, and access of all managers. DDM servers are composed of objects from the subclasses of the class MANAGER. Collectively, these objects perform all DDM processing on a single source or target system.

A server does not have to include all of the managers specified in the DDM Reference.

### 2.5.4 Manager

The class MANAGER is a subclass of OBJECT, and it is the superclass for all objects that allocate and control the space, existence, and access to objects. A manager imposes structure or order on a collection of more primitive objects like character strings, names, records, classes, commands, and reply messages. Examples of subclasses of MANAGER are RDB (relational database), SQLAM (Structured Query Language Application Manager), and SECMGR (Security).

A manager object belongs to a class of objects that has the class MANAGER as its superclass. All manager classes have a manager level (MGRLVL) to identify the level of functions performed by the manager.

The class MANAGER defines that all managers have the following in common:

- All managers organize data composed of scalar or collection objects.
- Objects only exist within a data stream structure or within a manager.
- Objects can combine and interact within a manager to form structures. For example, a variety of objects are required to define a command.

The following sections detail all the managers that can be part of a server as defined in the DDM Reference, including:

- Communications Manager
- Agents
- Dictionary
- Security Manager
- Supervisor
- RDB Manager
- SQL Application Manager

- Data Conversion
- Sync Point Manager
- XA Manager

#### 2.5.4.1 *Communications Manager*

Communications managers on the source system perform functions that differ from those performed by communications managers on the target system. One of the primary functions of communications managers on the source system is to interface with the system's communications facility for remote communications.

The communications manager accepts commands, replies, and objects (data) from an agent for transmission. The communications manager then provides the interface between DDM and the local communications facility by packaging each item it receives into the proper Data Stream Structure (DSS).

Data stream structures can be tied together in a process called chaining.

For each command received from one of its agents, a source communications manager builds a request data stream structure (RQSDSS) and places the command in it. The communications manager also generates a request correlation identifier, places it in the DSS, and returns the correlation identifier to the agent. A request correlation identifier associates a request with the request data, the replies to the request, and the data returned for that request.

The value of the request correlation identifier is a unique non-negative binary number. Each RQSDSS in a DSS chain must have a unique correlation identifier. The correlation identifier is sent to the target agent that receives the request.

For each object received from one of its agents, the communications manager builds an object data stream structure (OBJDSS) and places the object in it. The communications manager also places the correlation identifier of the associated RQSDSS in the OBJDSS, using the correlation identifier supplied by the agent. More than one object can be placed in the same OBJDSS if all of the objects have the same correlation identifier and there are no intervening reply messages.

For each reply message received from one of its agents, the target communications manager builds a reply data stream structure (RPYDSS) and places the reply in it. The communications manager also places the correlation identifier of the associated RQSDSS in the RPYDSS. The agent supplies the correlation identifier. More than one reply can be placed in the same RPYDSS if all of the replies have the same correlation identifier and there are no intervening reply data objects.

Requests, replies, and objects are transmitted exactly as they are presented to the communications manager.

#### 2.5.4.2 *Agents*

Agents on the source system perform functions that differ from those performed by agents on a target system. One of the primary functions of agents on the source system is to interface with the communications manager for required remote communications.

To access a relational database (RDB), an agent requests the communications manager to establish communications with an agent on the target system associated with the target server that manages the requested RDB. All requests for the file or RDB are directed through the source agent to the target agent.

#### 2.5.4.3 Dictionary

A dictionary is an object that contains the class descriptions of objects. Every dictionary entry is an object that has a name. For many entries there is also a codepoint that serves as an alias for the name in DDM data stream structures. No specific commands or protocols for dictionaries are provided in DDM.

The dictionaries describe objects in the DDM architecture. When an object is being processed, its validity is checked by referencing its codepoint in the proper dictionary.

The codepoint of every object specifies which DDM dictionary should be checked. The agent that has received the object uses the specified dictionary to validate that object.

#### 2.5.4.4 Security Manager

The security manager on the target system ensures that the requester is allowed access to only those files, commands, dictionaries, directories, or other objects for which authorization has been granted.

The security manager mechanisms can be defined in DDM or can use the communications manager built-in security mechanisms. All requester identification and verification must be performed by the system's local communications facilities or accessing the DDM security manager, negotiating the DDM authentication mechanism to use, and performing the security check.

For the relational database (RDB) model, the RDB manager determines a user's authorization to access or alter data or to administer the database. The security function of the RDB manager is not explicitly defined in DDM. However, information regarding RDB authorization errors is returned to the user in the SQL communications reply data (SQLCARD) object.

#### 2.5.4.5 Supervisor

A supervisor manages a collection of managers in a consistent manner. Within the server, the supervisor performs the functions that are server-oriented.

Supervisors provide an interface to their local system's services, such as file management, directory, dictionary, and security services.

DDM architecture defines only one command for supervisors, the exchange server attributes (EXCSAT) command. This command allows source and target servers to determine their respective server class names and levels of DDM support.

#### 2.5.4.6 RDB Manager

A relational database (RDB) is a database in which data is stored as a collection of tables (relations). The RDB manager is the program that accesses, locks, queries, and modifies an RDB. No specific commands for RDB managers are provided in DDM architecture.

Requests for RDB services can originate from an application program or an application user. The requests are routed from a Structured Query Language Application Manager (SQLAM) on the source system to an SQLAM on the target system. The RDB manager receives the requests from the target SQLAM, and returns RDB data to the target SQLAM for routing back to the requester.

Among other things, an RDB may contain the following objects:

- A catalog, which contains information about data, storage, collections, packages, and authorizations

- Tables that contain data organized in columns and rows
- Indexes, which are ordered sets of pointers to the data in relational tables
- Views, which provide alternate ways of looking a data in tables
- Locks, which prevent one program from accessing data that another program has changed but has not yet committed to the database
- Recovery logs, which contain information necessary for commit and rollback functions
- Packages, which define how an application program interacts with the RDB
- Authorizations, which define a user's rights to perform certain database functions
- Cursors, which allow an application program to point to a row of interest
- RDB collections, which are named, user-defined collections of packages

#### 2.5.4.7 *SQL Application Manager*

Structured Query Language (SQL) is the language used to access, query, and modify data in a relational database (RDB). An SQL Application Manager (SQLAM) provides a consistent way for application programs to submit SQL requests for RDB services.

An application program submits requests for RDB services to an SQLAM on the source system. The source SQLAM uses agent and communications manager services on both the source and target systems to route the request to a target SQLAM. The target SQLAM submits the requests to the RDB manager and returns RDB data back to the source SQLAM in the form of reply data objects and reply messages.

#### 2.5.4.8 *Data Conversion*

All data stored in an RDB or transferred between application programs and an RDB is described in terms of the data types defined by SQL. These data types are represented differently in different systems. Therefore, it is necessary to convert data as it is transferred between different servers.

Data conversion is performed by the target SQLAM for data the target SQLAM receives, and it is performed by the source SQLAM for data the source SQLAM receives. The source and target SQLAMs exchange the type to representation specifications during ACCRDB (access RDB) command processing.

#### 2.5.4.9 *Sync Point Manager*

DDM also helps to ensure that data will not be accidentally destroyed or altered, and that updates will not be lost because of conflicts between concurrent users.

Note that for relational databases, locking is an internal concern of the database system, but if the relational database system decides to support level 7, then they must ensure that a set of DRDA protected connections can share resources, so as to prevent deadlocks from occurring. However, all of the data integrity functions provided by a relational database product can be fully utilized.

Both one phase and two-phase commit processing, respectively used for single and multiple site updating, are documented in the DDM architecture which is based on the SYNCPTMGR (Sync Point Manager) and RSYNCPTMGR (Resynchronization Manager). Transactions against the relational database are gathered in Units of Work (UOW), and the resource recovery process runs at the UOW level. All the activity within an UOW is committed (or backed out) and then a new UOW is started.

#### 2.5.4.10 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)  
Obtain a list of Prepared and Heuristically completed Transactions at the application server.
7. SYNCCTL(Forget)  
Ask 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. 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.

Transactions are associated with an XID. A valid XID represents a Global Transaction with one or branches, while a NULL XID represents a Local Transaction. The application is responsible for protecting all Global Transactions using the two-phase protocol, but must drive local commit or rollback for Local Transactions. The XAMGR expects the RDB to provide resource sharing capabilities for tightly-coupled and loosely coupled XA connections. For more details, see *XAMGROV* (on page 1066).

#### 2.5.5 Connectivity

This section describes how the DDM architecture can be implemented to allow remote systems to exchange data regardless of the compatibility of the software or hardware of the systems.



### 2.5.5.1 Data Connectivity

Data connectivity is the ability to share data between two systems. Data connectivity is accomplished through the canonical representation of data objects, commands, and replies. These canonical representations are placed in data stream structures for transmission over the local communications facilities. DDM data stream structures can be transmitted and received by using many different local communications facilities.

To provide data connectivity between different products, the DDM architecture defines only a limited number of DDM protocols. DDM protocols are designed for specific communications environments, and they define DDM communications in those environments. Products implementing DDM select the environments (and thus the protocols) they want.

Data connectivity is achieved through the following features of the DDM architecture:

- Negotiation
- Selecting Subsets
- Developing Product-Unique Extensions

### 2.5.5.2 Negotiation

After communications have been established with the target system, a source system that desires data connectivity using the DDM architecture must negotiate or exchange the information it needs to attain data connectivity with the target system.

#### **EXCSAT Command**

Once communication has been established, negotiation begins with the exchange server attributes (EXCSAT) command. This command exchanges the following information between servers:

- The server's class name
- The architectural level of each class of managers the server supports
- The server's product release level
- The external name of the job, task, or process associated with the server

The source system's server sends the EXCSAT command to the target server to identify itself to the target server, and to inquire which managers the target server supports. The target server responds by returning a server attributes reply data object (EXCSATRD) to identify itself and the managers it supports. These must be the first commands exchanged between the source and target systems. Any other opening exchange causes a conversational protocol error (PRCCNVRM) to be returned.

#### **Exchanging Server Class Names**

Servers are classified by the type of hardware or operating system that provides their local processing environment. Servers exchange server class names so that each server can determine which code point dictionaries are available. The DDM architecture assigns each codepoint dictionary a fixed index value in the list of dictionaries known to all servers. The various classes of servers need not support all dictionaries. Server class name is a required parameter of the EXCSAT command.

### **Manager-Level Support**

Each manager class defined by DDM has a manager-level number (MGRLVLN) variable defined in its class description. As the definition of a manager class in the DDM architecture evolves, higher manager levels are sequentially assigned to its MGRLVLN variable.

The DDM architecture requires that higher manager levels support all functions and capabilities of the lower manager levels, but the inverse need not be true. For example, a security manager with an MGRLVLN equal to three must support all the functions and capabilities of MGRLVLN two and MGRLVLN one security managers. However, MGRLVLN one security managers are not required to support all functions and capabilities of MGRLVLN two or MGRLVLN three security managers.

When a source server sends an EXCSAT command, it optionally specifies the manager-level list (MGRLVLLS) parameter. Each entry in the MGRLVLLS consists of the codepoint of a manager class and the level of support requested by the source server. When the target server receives an EXCSAT command, it examines each of the entries and returns a MGRLVLLS parameter in the EXCSATRD that indicates its level of support for each of the manager classes specified by the source server. The target server must not provide any information on any target managers unless that information is explicitly requested by the source server.

If the target server's support level for a manager class is greater than or equal to the source server's level, the source server's level is returned for that class, providing the target server can operate at the source's level; otherwise, a level of 0 is returned. If the target server's support level is less than the source server's level, the target server's level is returned for that manager class. If the target server does not recognize the codepoint of a manager class, or does not support that manager class, it returns a support level of zero. The target server then waits for the next command, or for termination of communications by the source server.

When the source server receives the EXCSATRD, it compares each of the entries in the MGRLVLLS it received to the entries in the MGRLVLLS it sent. If there is any mismatch in the levels, the source server determines if it can use or adjust to the lower level of target support for that manager class. Or, the source server can decide to terminate communications. DDM architecture does not explicitly define any criteria for making this decision. The source server can also attempt to use whatever commands are requested by its user and rely upon receiving a command not supported reply message (CMDNSPRM) for mismatches.

### **Server Product Release Level**

The source server optionally sends the server product release level (SRVRLSLV) parameter with its EXCSAT command to provide the target server with information about the source's product release level. The target server optionally returns the SRVRLSLV parameter on the EXCSATRD to provide the source server with information about the target's product release level.

Since server product release levels are defined as strings that are not explicitly defined in the DDM architecture, this information is likely to be of use only when the source and target server have the same server class name. The DDM architecture does not explicitly define any use for this information, and it can be ignored by either the source or the target server.

#### *2.5.5.3 Selecting Subsets*

The goal of DDM is to maximize data connectivity among products implementing the architecture. However, DDM does not:

- Require all products implementing the DDM architecture to support all of the DDM architecture functions and associated commands

- Require all products implementing the DDM architecture to support the relational database model and the Structured Query Language Application Manager (SQLAM) and its associated commands
- Allow products implementing the DDM architecture to arbitrarily select the commands that exactly match the capabilities of their local data management system
- Restrict products implementing DDM to only the objects defined in the Reference

Therefore, DDM includes rules for selecting subsets of the DDM architecture, and for developing product-unique extensions.

Each DDM subset consists of a number of classes of objects and the commands to which they respond. However, even with a subsetting policy, an exact match may not exist between DDM subsets and a system's local data management. In this case, it may be necessary for the products implementing DDM to:

- Emulate DDM commands on a target system by using a sequence of target data management requests
- Emulate source data management functions on a target system by sending a chain of DDM commands

### Subsets Selection Rules

The following rules have been adopted for the selection of subsets of DDM architecture. The implementing programmer must:

1. Select the server class to be implemented.
2. Select the communications manager to be supported (SNA or TCP/IP).
3. Select the DDM agent class to be supported

DDM agents represent the requester on both the source system and the target system. The source agent is bound to a target agent to actually request services from the target and return replies to the source system. One class of agent is presently defined.

Additional DDM support is not required if a server acts only as a source of requests (source server) and does not provide remote data management services for remote requesters (target server).

4. All target servers must support the following DDM manager classes:
  - Supervisor
  - Security manager
  - Directory
  - Dictionary

5. Select the DDM relational database (RDB) class to be supported.

RDB is the only relational database class defined. It defines how data in an RDB is accessed and modified.

A relational database product implementing DDM must:

- Support all commands identified as REQUIRED in the DDM architecture for the Structured Query Language Application Manager (SQLAM) class.

- Support as many of the OPTIONAL commands of the SQLAM classes as possible to enhance connectivity with other products.

### **Unsupported Commands, Parameters, Values, and Objects**

DDM architecture requires each target server to reply to unrecognized and unsupported commands, parameters, parameter values, and command objects with one of the following reply messages:

- CMDNSPRM (command not supported)
- PRMNSPRM (parameter not supported)
- VALNSPRM (parameter value not supported)
- OBJNSPRM (object not supported)

#### *2.5.5.4 Developing Product-Unique Extensions*

Existing DDM classes, commands, parameters, and messages can be used in product extensions to DDM, intermixed with product-defined structures as needed. Some extensions will be unique to a particular product.

The following requirements must be met to develop product extensions:

- All products implementing DDM must support data connectivity with other products implementing DDM based solely on the codepoints in the dictionaries from the DDM Reference. While agreements between products to support extensions are permitted, they must not be required.
- Product extension codepoints are sent only if both the source and the target server have the product extension dictionary in their dictionary list.

The framework of existing DDM classes can be used as the basis for extensions to the DDM architecture. DDM allows the following open architecture enhancements:

- New classes of objects, including new commands and replies unique to the class, or using existing DDM commands when appropriate
- New commands for existing DDM classes
- New parameters for existing DDM commands
- New values for existing DDM parameters

Products must assign codepoints to their own product-defined extensions that do not conflict with the codepoints of the DDM architecture.

### **2.5.6 DDM Object-Oriented Programming**

DDM is based on the concepts of object-oriented programming. This approach fundamentally differs from traditional programming in a number of ways.

Traditional programming languages are comprised of operations which perform functions on operands. Operations are viewed as the active parts of a language. Operands are viewed as passive, and change only when an operation acts on them.

The action of an operation on an operand is determined by the environment. The environment can be the computer user, an application program, or anything else that affects an operation. Therefore, operations and their operands are treated as independent entities.

Object-oriented programming is a step towards breaking operations' dependence upon environment. Object-oriented programming replaces the operands and operations concept with objects and commands.

Objects are both data and the operations that can be performed on that data. Commands are requests to an object asking it to perform one of its operations.

An operation is initiated by sending a command to an object telling it what to do. The object, itself, determines how to actually carry out the operation by selecting the command implementation from a table of commands it supports. The environment is no longer responsible for this part of the operation.

DDM is documented through an object-oriented approach to programming. The rest of this chapter defines the main parts of DDM and how they fit within object-oriented programming. Then, the concept of using DDM as a data connectivity language is explored.

### 2.5.6.1 Object

All DDM architecture can be broken down into small, well-defined, and standardized units called objects. Objects are data processing entities. All objects are composed of a contiguous string of bytes with a specific format that describes the object. This format is composed of:

- Length
- Codepoint
- Data

The length is a 2-byte binary value that contains the length of the object. The length value is always the first part of any object. The value of the length field includes:

- Length of the length field
- Length of the codepoint field
- Total length of the object's data

The codepoint is a 2-byte hexadecimal value that indicates where the class description of the object can be found in a dictionary.

The data area of an object is composed of one or more contiguous bytes. These bytes may contain the following:

Scalars      Byte strings over which one or more data values have been mapped.

Collections    Objects that contain one or more objects in their data area.

A simple scalar is a scalar with only a single data value. Examples of simple scalars are binary numbers, character strings, and Booleans. A mapped scalar is a scalar with multiple contiguous data values.

### 2.5.6.2 Class

Objects are described by objects called classes. A class is an object that describes a set of objects that have a common structure and that respond to the same commands. Every class description includes:

- A description of the structure of the data area of the object
- A list of the specific commands to which the object can respond

- A list of specific responses for the object
- A description of the function of the object

The types of operations that can be performed on members of a class are defined by the class, itself. Classes are themselves objects, and they are self-describing; that is, they are described by the class CLASS.

An operation is performed by sending a command to an object telling it what to do. The object itself actually determines how to carry out the operation by selecting the command implementation from a table of commands supported by its class.

The DDM architecture does not provide an explicit definition of the complete structure and protocol of all classes.

### 2.5.6.3 Class Inheritance

Class inheritance is used by DDM architecture to ensure reusability, consistency, accuracy, and modularity. Every structure in DDM is composed of new classes built or derived from old classes. The new class maintains all of the variables and behaviors of the old class, while adding its own behaviors and variables. Therefore, new classes become increasingly specialized.

A subclass of a class is a new class derived from an old class. The new class has all of the variables of the old class, and can perform all the functions of the old class. But it adds variables and functions of its own.

### 2.5.6.4 Grammar

A grammar describes all of the rules for building words into bigger structures. The primary grammatical structures of DDM are:

- Commands
- Reply messages
- Control messages
- Reply data stream structures (RPYDSS)
- Request data stream structures (RQSDSS)
- Object data stream structures (OBJDSS)

All of these structures must be built according to certain rules.

### 2.5.6.5 Protocols

A protocol is the method a computer system uses to communicate. Different computer systems have different communications protocols. DDM data stream structures can be transmitted and received through many different communications facilities, all using the same protocol.

The goal of DDM is to ensure that computer systems of different implementations can communicate. To this end, a limited number of DDM protocols are defined as part of the DDM architecture. DDM protocols are designed for a specific communications environment. Products implementing DDM can select the environments (and thus the protocols) to be supported.

It is important that the target communications manager and the source communications manager take turns sending and receiving data. This provides for an orderly exchange of information between two systems. If this orderly protocol is not followed, information can be lost or damaged, and the two systems will not be able to communicate successfully.

The communications manager whose turn it is to send data structures has the right to send for the conversation. The sending communications manager then passes the right to send for the conversation to the receiving communications manager.

To ensure the orderly exchange of data, requests, and replies, the following DDM conversational protocols must be obeyed:

- The source communications manager must not send a reply data stream structure (RPYDSS) to the target communications manager.
- The source communications manager can only send one request data stream structure (RQSDSS), or a chain of RQSDSSs according to the chaining rules. Then, the source communications manager must wait for a reply data stream structure (RPYDSS), or an object data stream structure (OBJDSS) from the target communications manager.
- The target communications manager must not send an RQSDSS to the source communications manager.
- The target communications manager can only send one RPYDSS, an RPYDSS chain, or an OBJDSS. Then the target communications manager must wait for an RQSDSS or RQSDSS chain from the source communications manager.
- The first RQSDSS sent by the source communications manager must contain an exchange server attributes (EXCSAT) command.

The source communications manager, once communications have been established, has the right to send to the target communications manager. A source agent can now request the source communications manager to send commands to the target agent and to pass the right to send. The source communications manager then waits for the target to send an RPYDSS, RPYDSS chain, OBJDSS, or OBJDSS chain, and return the right to send.

When the target agent receives a command, it executes the command. Command replies are passed to the target communications manager where they are put in a queue. When the target communications manager has the right to send, it sends all replies to the source communications manager and passes the right to send for the conversation to the source communications manager.

The basic pattern of the source agent sending commands/data and the target agent sending replies/data is repeated until the user no longer needs remote data management services from the target system.

### 2.5.7 DDM Dictionaries

The dictionaries that form the DDM Reference are:

- The DDM Tutorial Objects Dictionary
- The DDM Primitive Classes Dictionary
- The DDM Base Classes Dictionary
- The DDM Relational Database Classes Dictionary

Each dictionary defines a unique set of objects. However, all the defined terms from all the dictionaries are merged alphabetically to formally describe DDM. Dictionary entries are often defined through other objects that are defined elsewhere within the dictionaries. A label near the top of the first page of each description indicates which dictionary contains the described term.

Each term is described in an entry that can be referred to as a *term page* (much like the form of a UNIX manual page). Each term page contains the following information:

1. The term's DDM name and a short description of the term.
2. The dictionary in which the term is found and the codepoint, if any, for the term.

Each dictionary also has a DDM abbreviated term name. The dictionary abbreviations are:

QDDTTRD DDM Tutorial Objects Dictionary (codepoint: NONE)

QDDPRMD DDM Primitive Classes Dictionary (codepoints in X'0000' range)

QDDBASD DDM Base Classes Dictionary (codepoints in X'1nnn' range)

QDDRDBD DDM RDB Classes Dictionary (codepoints in X'2nnn' range)

3. The remainder of the term definition contains the variables that provide information about the term.

All objects of the same class have the same variables. Terms in the dictionaries have the following variables as part of their definition: Length, Class, and Description (Semantic). The information given in these sections is as follows:

Length        The length of the object.

Class         The name of the class to which the term belongs, a short description of the class name, and the dictionary in which the class is defined.

Description (Semantic)

A description of the class and information common to all instances of the class.

This section often contains examples and diagrams to further explain the class.

Term definitions may also contain additional variables, depending on the class to which the term belongs.

4. The See Also list follows the term description.

#### 2.5.7.1 Dictionary Contents

Each definition of an object in the dictionaries consists of named variables. These variables divide the term definitions into sections that contain information about the term.

Dictionary objects are arranged in alphabetical order. However, the variables of an object often refer to other objects in the same or another dictionary. This creates a second type of organization, in addition to alphabetical, within the dictionaries. The objects form a hierarchical structure from the most abstract (generalized), such as the object DDM, to the least abstract, primitive data.

#### 2.5.7.2 Tutorial Objects Dictionary

The DDM Tutorial Objects Dictionary, or tutorial dictionary, contains Menu and Help objects that provide information about DDM. The objects in the tutorial dictionary describe concepts and structure that provide a basic understanding of DDM. After the reader has studied the ABBREVIATIONS description and begun to understand the meanings of the abbreviated names, the next logical step is to study the DDM section. The menu in that section will lead to a deeper understanding of DDM architecture.

The terms in the tutorial dictionary are instances of the class HELP, or the class MENU, and provide basic information about DDM.



### 2.5.7.3 *Primitive Classes Dictionary*

The DDM Primitive Classes Dictionary, or primitive dictionary, contains definitions of all DDM classes that describe the primitive data objects used as the descriptive foundation for DDM. The Primitive Classes Dictionary (and also the Base Classes Dictionary) must be supported by every product implementing DDM architecture. The products do not, however, have to implement the dictionaries or class descriptors as they are documented in the architecture; only the semantic equivalents of the dictionaries and classes are required to be implemented.

The terms in the primitive dictionary describe the primitive data objects upon which DDM is based. Most of the terms in this dictionary are instances of the class CLASS. The term descriptions provide the information necessary to implement DDM.

Definitions of objects in the class CLASS include the variables class, title, status, and help, but also have additional variables that supply more information. Terms of the class CLASS always have the variable SPRCLS (superclass). The superclass is the class from which the term class inherits its characteristics. The superclass is a higher-level, more abstract object than the class.

Terms whose class is CLASS also have the following variables (the DDM name is given in parentheses):

**Class variable (CLSVAR)**

Defines the variables of the class, but not instances of the class.

These variables represent the structure of the class, or contain information common to all instances of the class.

**Class command (CLSCMD)**

Defines a set of commands which describe the operations that can be performed by the class.

Typical operations are instance creation and initialization.

**Instance variable (INSVAR)**

Defines the variables specific to an instance of the class.

**Instance command (INSCMD)**

Defines a set of commands that can be performed by instances of the class.

Terms may also have additional variables which further define the class.

### 2.5.7.4 *Base Classes Dictionary*

The DDM Base Classes Dictionary, or base dictionary, contains all DDM classes that describe the data objects for DDM. Like the Primitive Classes Dictionary, the Base Classes Dictionary must be supported by every product implementing DDM architecture.

### 2.5.7.5 *Relational Database Classes Dictionary*

The DDM Relational Database Classes Dictionary, or RDB dictionary, contains all DDM classes that describe the relational database objects for DDM. Every product implementing the DDM architecture that requires relational database services must support the RDB dictionary.

### **2.5.8 Using the DDM Reference Manual**

When the reader first begins to study the DDM architecture many of the variables of a term can be ignored. Concentrating on the semantic variables will give a general idea of the purpose and use of each term.


Begin by reading the OVERVIEW and INHERITANCE terms to obtain a broad description of the DDM concept. Many of the terms in the tutorial dictionary give more specific information on the terminology of DDM.

To go further into a particular DDM term, refer to the term named by the Class variable. This gives more general information about the characteristics of the term.

The hierarchical design of DDM allows the reader to learn the language in a systematic manner. The progression through the DDM Reference should be:

1. Read the text of a term for a general description.
2. Read the names and descriptions of all referenced terms.
3. Go to the referenced terms and read their text.
4. Read the variable details of those referenced terms which are unfamiliar.

Regarding the steps noted above, it is helpful to number a set of bookmarks to use as place markers as the reader progresses from term to term in the document.



*Chapter 3*  
***Terms***

This chapter contains the DDM term pages arranged alphabetically.

**NAME**

ABBREVIATIONS — DDM Dictionary Abbreviations

**DESCRIPTION (Semantic)**

**Dictionary** QDDTTRD

**Length** \*

**Class** HELP

DDM Dictionary Abbreviations (ABBREVIATIONS) describes the methods for creating abbreviations for terms in the DDM architecture and contains a list of these abbreviations.

Formulating meaningful names for variables, commands, and programs is one of the more difficult aspects of programming. In the DDM dictionary, the names of the terms have been formulated by consistently applying the following rules:

1. Single words can be used without abbreviation, such as TRUE or FILE.
2. Abbreviations are three letters long and consist of the first letter of the word followed by the next two consonants (such as CRT for create), or the next two letters for words with fewer consonants or a commonly accepted abbreviation (such as DEL for Delete).
3. Phrases of two words consist of two three-letter abbreviations such as BGNBND for Begin Bind.
4. Longer phrases consist of an abbreviation of the first word, optionally an abbreviation of the next word, and the first letters of additional words, such as CMDNSPRM for Command Not Supported Reply Message.
5. Command name abbreviations begin with a verb, followed by the class of the object operated on, such as ACCRDB for the Access RDB command.
6. Object attributes begin with the object class name followed by the attribute name, such as FILCLS for file class.
7. The same terminology is used for all instances of the same group; for example, all reply messages end in RM, and all access method names end in AM.

The two and three-character abbreviations commonly used in term names are as follows:

<b>Abbreviation</b>	<b>Meaning</b>
ABN	Abnormal
ABS	Absolute
ACC	Access
ACT	Active
ADR	Address
AGN	Agent
AI	Access intent
AL	Attribute list
ALC	Allocate
ALT	Alternate

<b>Abbreviation</b>	<b>Meaning</b>
ALW	Allow, allowed
AM	Access method
AMT	Amount
AP	Apostrophe
APP	Advanced-Program-to-Program, append, apply
ARC	Archive
ASC	Ascending, ASCII
ASG	Assign
ASM	Assumptions
ASY	Asynchronous
ATH	Authorized, authorization
AT, ATT	Attribute
AVI	Already verified indicator
AVL	Available
BAS	Base
BCK	Back
BFF	Buffers
BGN	Begin
BIN	Binary
BK	Book
BLK	Block
BNA	Bind not active
BND	Bind
BOF	Beginning-of-file
BPA	Bind process active
BRK	Broken
BYP	Bypass
BYT	Byte
CCS	Common Communications Support
CD	Current directory
CD, COD	Code
CHG	Change
CHK	Check
CHN	Chaining
CLR	Clear
CL, CLS	Class, close, classified
CMA	Comma
CMB	Combined
CMD	Command
CMM	Commit

<b>Abbreviation</b>	<b>Meaning</b>
CMN	Communications
CMP	Complete, completed, compatibility
CN, CNT	Content, count
CNF	Conflict
CNL	Cancel
CNS	Constant, consistency
CNT	Count, continue, content
CNV	Conversational
COL	Collection
CP	Capability
CPY	Copy
CRR	Correlation, current
CR, CRT	Create
CS	Cursor stability
CSN	Consistency token and section number
CSR	Cursor
CST	Character subtype
CT	Consistency token
CTL	Control
DAT, DT	Date
DBC	Double byte character
DCE	Distributed Computing Environment
DCL	Declare
DCT	Dictionary
DDM	Distributed Data Management
DEC	Decimal
DEF	Definition
DEL	Delete, delimiter
DEP	Dependency
DF	Definition
DFT	Default
DGT	Digit
DGN	Diagnostic
DIR	Direct
DMG	Damage
DO	Duplicate option
DQ	Double quote
DR	Data representation
DRC	Directory
DRN	Directory name

<b>Abbreviation</b>	<b>Meaning</b>
DRP	Drop
DRT	Dirty
DSC	Descending, describe, descriptor
DSP	Displacement, disposition
DT, DTA	Data
DUP	Duplicate
EF, EOF	End-of-file
ELM	Element
EMP	Empty
EN	Entry
ENC	Encoding
ENU	Enumerated
ER, ERR	Error
ETH	Either
EUR	European
EX, EXN	Extent
EXC	Exclusive, exchange, execute
EXP	Expiration, expected, explain
EXS	Existing, exists, existence
EXT	External
F, FIL	File
FAT	File attributes
FB	Feedback
FCT	Factor
FDO	Formatted Data: Object Content Architecture
FIX	Fixed
FL	Failure
FLD	Field
FLP	Floating point
FM, FMT	Format
FML	Family
FN, FND	Find, found
FNV	Format not valid
FR, FRS	First
FRC	Force
FRG	Forget
FUL	Full
HDL	Handle
HDR	Header
HLD	Hold

<b>Abbreviation</b>	<b>Meaning</b>
HRR	Hierarchical
ID	Identifier
IMM	Immediate
IN	Initialization
INA	Inactive
IND	Index, indicator
INI	Initial
INS	Insert
INT	Intent, interrupt
INV	Invalid
ISO	International Standards Organization, isolation
IUS	In use
JIS	Japanese Industrial Standard
KP	Keep
KY	Key
LCK, LK	Lock
LEN	Length
LFT	Left
LL	Length (encoded)
LM, LMT	Limit, limits, limited
LO	Lock option
LOC	Location
LOD	Load
LRG	Large
LS, LST	List, last
MAX	Maximum
MBR	Member
MCH	Match, mismatch
MEC	Mechanism
MGM	Management
MGR	Manager
MIN	Minimum
MIX	Mixed
MLV	Multi-leave
MOD	Modify
MNS	Minus
MSG	Message
MST	Must
MTH	Method
MTL	Mutually



<b>Abbreviation</b>	<b>Meaning</b>
MVE	Move
NAC	Not accessed
NAM, NM	Name
NEM	Not empty
NB, NBR	Number
NER	No error
NFN	Not found
NGT	Negotiate
NON	None
NOP	Not open
NSP	Not supported
NUM	Numeric
NX, NXT	Next
OBJ	Object
OFF	Offset
OLO	Open lock option
ONL	Only
OP, OPT	Option
OPN	Open
OPR	Operator, operation
ORD	Order
OUT	Output
OVR	Overview
OWN	Owner
PGM	Program
PKG	Package
PLS	Plus
PNT, PT	Point
POS	Position
PR	Processing, previous
PRC	Protocol, process, processing, precision
PRD	Period
PRF	Profile
PRG	Program
PRM	Permanent, parameter
PRP	Prepare
PRT	Protected
PRV	Previous
PS	Position
PWD	Password

<b>Abbreviation</b>	<b>Meaning</b>
PTT	Partitioned
PVL	Privilege
QLF	Qualified
QRY	Query
QUE	Queue
RD	Reply/request data, read
RBK	Rollback
RCV	Receive, recoverable
RDB	Relational database
REC	Record
REF	Reference
REG	Register
REL	Relative, relational
RFR	Refresh
RFM	Record format
RGH	Right
RL	Request list, record length
RLL	Roll
RLS	Release
RM	Reply message
RNA	Resource not available
RNB	Record number
RND	Random
RNM	Rename
RPL	Replace
RPY	Reply
RQS	Request
RR	Repeatable read
RSC	Resource
RTN	Return, retention, retain
RUL	Rule, rules
RVK	Revoke, revoked
SAT	Server attributes
SBC	Single-byte character
SBM	Submit
SBS	Subset
SCC	Successfully
SCL	Scaling
SCM	Source communications manager
SCR	Scrolling

<b>Abbreviation</b>	<b>Meaning</b>
SCT	Section
SDA	Scalar data array
SEC	Security
SEQ	Sequence, sequential
SES	Session
SGN	Signed
SHD	Shadow
SHR	Share
SIZ	Size
SN	Section number
SNA	Space not available, Systems Network Architecture
SND	Send
SNG	Single
SP	Supported
SPC	Space, special
SPR	Super
SPV	Supervisor
SQL	Structured Query Language
SRC	Source
SRV	Server
ST	Status
STG	Storage
STP	Stop
STR	String, stream
STT	Statement
SVR	Severity
SYM	Symbol
SYN	Syntax, synchronous
SYNC	Sync point
SYNCPT	Sync point
SYS	System
SWT	Switch
SZ	Size
TBL	Table
TCM	Target communications manager
TIM	Time
TKN	Token
TMP	Temporary
TNA	Temporarily not available
TP	Transaction program

<b>Abbreviation</b>	<b>Meaning</b>
TPN	Transaction program name
TRG	Target
TRN	Truncation
TRP	Triplet
TTL	Total
TXT	Text
TYP	Type
ULD	Unload
UNK	Unknown
UNL	Unlock
UNN	uninterrupted
UNT	units
UOW	unit of work
UPD	update, updater
USR	user
VAL, VL	Value
VAR	Varying
VRB	Variable
VRS	Version
VLT	Violation
WRN	Warning
WRT	Write
XFR	Transfer
XLT	Translate table
XMS	Transmission
ZON	Zone

**SEE ALSO**

**Semantic**

*CONCEPTS* (on page 243)

**NAME**

ABNUOWRM — Abnormal End Unit of Work Condition

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'220D'  
**Length** \*  
**Class** CLASS  
**Sprcls** RPYMSG - Reply Message

Abnormal End Unit of Work Condition (ABNUOWRM) Reply Message indicates that the current unit of work ended abnormally because of some action at the target server. This can be caused by a deadlock resolution, operator intervention, or some similar situation that caused the relational database (RDB) to rollback the current unit of work. This reply message is returned only if an SQLAM issues the command.

Whenever an ABNUOWRM is returned in response to a command, an SQLCARD object must also be returned following the ABNUOWRM.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'220D'	
<b>rdbnam</b>	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
<b>svrdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**cmdrpy** BGNBND (on page 110)  
 BNDSQLSTT (on page 136)  
 CLSQRY (on page 165)  
 CNTQRY (on page 222)  
 DRPPKG (on page 293)  
 DSCRDBTBL (on page 300)  
 DSCSQLSTT (on page 304)  
 ENDBND (on page 336)  
 EXCSQLIMM (on page 371)  
 EXCSQLSET (on page 377)  
 EXCSQLSTT (on page 381)  
 OPNQRY (on page 555)  
 PRPSQLSTT (on page 636)

**Semantic**

*RDBCMM* (on page 728)  
*REBIND* (on page 753)  
*DSCRDBTBL* (on page 300)  
*DSCSQLSTT* (on page 304)  
*PRPSQLSTT* (on page 636)

*CNTQRY* (on page 222)  
*DSCRDBTBL* (on page 300)  
*DSCSQLSTT* (on page 304)  
*DSS* (on page 308)  
*LMTBLKPRC* (on page 475)  
*PRPSQLSTT* (on page 636)  
*QRYNOPRM* (on page 690)  
*SYNCPTOV* (on page 944)

**NAME**

ACCDMG — Access Damage Severity Code

**DESCRIPTION (Semantic)****Dictionary** QDDPRMD**Codepoint** X'003E'**Length** \***Class** CONSTANT

Access Damage Severity Code (ACCDMG) specifies that the target agent's ability to access a relational database (RDB) has been damaged.

The following steps are required to recover an RDB:

1. Terminate communications with the target server.
2. Re-establish communications with the target server.
3. Reaccess the RDB.

---

value	3
-------	---

**SEE ALSO**

**insvar**      *AGNPRMRM* (on page 65)  
                  *CMDCHKRM* (on page 173)  
                  *RSCLMTRM* (on page 778)  
                  *SVRCOD* (on page 911)

## NAME

ACCRDB — Access RDB

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD**Codepoint** X'2001'**Length** \***Class** CLASS**Sprcls** COMMAND - Command

Access RDB (ACCRDB) command makes a named relational database (RDB) available to a requester by creating an instance of an SQL application manager. The access RDB command then binds the created instance to the target agent and to the RDB. The RDB remains available (accessed) until the communications conversation is terminated.

**Source System Processing**

The source system determines the location of the RDB:

Local            Call the local RDB server.

Remote          Send the ACCRDB command to the remote RDB server.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergo conversion between the code pages.

**DSS Carrier: RQSDSS****Target System Processing**

The *armcorr* parameter specifies the correlator that is used to convey contextual information about all SQL requests for a connection in an environment with Application Response Measurement (ARM) instrumentation. If the *armcorr* value is not specified, then SQL requests are not subject to ARM instrumentation based on an ARM correlator.

The *crrtkn* parameter carries a token to correlate with the work for an application at the source and target servers. This token is used for alerts and other diagnostic activities.

The *prdid* parameter specifies the product release level of the source products accessing a remote relational database.

The *prddta* parameter carries product-specific information.

The *rdbaccl* parameter specifies an instance of the SQLAM that accesses the RDB.

The *rdbalwupd* parameter indicates whether the RDB allows the requester to perform any update operations in the RDB. If update operations are not allowed, the requester is limited to read-only access of the RDB resources.

The *rdbnam* parameter specifies the name of the RDB.

The *sttdeccl* parameter specifies the character used as the decimal delimiter in dynamic SQL statements.



The *sttstrdel* parameter specifies the characters used to delimit character strings and delimited SQL identifiers in dynamic SQL statements.

The *typdefnam* parameter specifies the name of the data type to data representation mapping definitions. These definitions are used when the source SQLAM sends command data objects for the commands which follow.

The *typdefovr* parameter specifies the single-byte, double-byte, and mixed-byte CCSIDs of the Scalar Data Arrays (SDA) in the identified data type to the data representation mapping definitions. The *typdefovr* parameter can contain three CCSIDs. The first occurrence of the ACCRDB command must contain the CCSIDSBC CCSID; it can optionally specify the other two CCSIDs.

The ACCRDBRM is returned after the RDB is successfully accessed.

The *DIAGLVLs* parameter controls the level of diagnostics requested in the SQL communications area:

**DIAGLVL0** A null SQLDIAGGRP is returned. This is the default.

**DIAGLVL1** A non-null SQLDIAGGRP should be returned.

Refer to the DRDA Reference for a description of the SQLDIAGGRP FD:OCA descriptors.

### Data Conversion

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergo conversion between the code pages.

### Exceptions

If the communications conversation is allocated with a synchronization level other than NONE, the CMNAPPC is being used, and the command target is SQLAM, the command is rejected with the MGRDEPRM.

If the communications conversation is allocated with a synchronization level other than SYNCPT, the CMNSYNCPT is being used, and the command target is SQLAM, then the command is rejected with the MGRDEPRM.

If, on the EXCSAT command, the SUPERVISOR manager has not received the information that the SQLAM needs, then the command is rejected with the MGRDEPRM.

If an RDB is currently accessed by the requester on the same conversation, then the command is rejected with the RDBACCRM.

If the requester is not authorized to access the specified RDB, then the command is rejected with the RDBATHRM.

If the specified RDB cannot be found, then the command is rejected with the RDBNFNRM.

If the target SQLAM cannot support the specified data type to data representation mapping definition (TYPDEFNAM), then the command is rejected with the VALNSPRM.

If the target SQLAM cannot support the single-byte CCSID specified by the *typdefovr* parameter, then the command is rejected with the VALNSPRM.

If the target SQLAM cannot support the double-byte or mixed-byte CCSIDs specified by the *typdefovr* parameter, then the ACCRDBRM with a severity code WARNING is returned.

For a successful database connection, an optional SQLCARD reply data object may be sent by the target server following the ACCRDBRM reply message for returning an SQL warning and/or server-specific connect tokens to the source server.

If the RDB is currently unavailable, then the RDBAFLRM is returned followed by an SQLCARD stating the reason for the unavailability, and the target SQLAM instance is terminated.

**Source System Reply Processing**

Return any reply messages and data objects to the requester.

**Handling Subsequent ACCRDB Commands**

The source server can send repeated ACCRDB commands when it desires to reuse a connection for another application or transaction. The ACCRDB must be preceded by an EXCSAT, ACCSEC, and SECCHK.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergo conversion between the code pages.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'2001'	
<b>armcorr</b>	INSTANCE_OF OPTIONAL IGNORABLE MINLVL	ARMCORR - ARM Correlator  7
<b>crtrkn</b>	INSTANCE_OF REQUIRED MINLVL	CRRTKN - Correlation Token † 4
<b>prddta</b>	INSTANCE_OF OPTIONAL IGNORABLE	PRDDTA - Product-specific Data
<b>prdid</b>	INSTANCE_OF REQUIRED	PRDID - Product-specific Identifier
<b>rdbacccl</b>	INSTANCE_OF ENUVAL REQUIRED CMDTRG	RDBACCCL - RDB Access Manager Class X'2407' - SQLAM - SQL Application Manager
<b>rdbalwupd</b>	INSTANCE_OF OPTIONAL	RDBALWUPD - RDB Allow Updates
<b>rdbnam</b>	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
<b>sttdecdel</b>	INSTANCE_OF	STTDECDEL - Statement Decimal Delimiter

	OPTIONAL	
sttstrdel	INSTANCE_OF OPTIONAL	STTSTRDEL - Statement String Delimiter
typdefnam	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL MINLVL ENUVAL MINLVL REQUIRED	TYPDEFNAM - Data Type Definition Name 'QTDSQL370' 'QTDSQL400' 'QTDSQLX86' 'QTDSQLASC' 4 'QTDSQLVAX' 4
typdefovr	INSTANCE_OF REQUIRED	TYPDEFOVR - TYPDEF Overrides
diagnostics	INSTANCE_OF MINLVL OPTIONAL	DIAGLVL - SQL Error Diagnostic Level 7
clscmd	NIL	
inscmd	NIL	
cmddta	NIL	
<b>rpydta</b>		<b>REPLY OBJECTS</b>
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL MINLVL ENUVAL MINLVL OPTIONAL MTLINC	TYPDEFNAM - Data Type Definition Name 'QTDSQL370' 'QTDSQL400' 'QTDSQLX86' 'QTDSQLASC' 4 'QTDSQLVAX' 4  X'2408' - SQLCARD - SQL Communications Area Reply Data
	NOTE	If RDBAFLRM is returned.
X'0035'	INSTANCE_OF OPTIONAL MTLINC	TYPDEFOVR - TYPDEF Overrides  X'2408' - SQLCARD - SQL Communications Area Reply Data
	NOTE	If RDBAFLRM is returned.
X'2408'	INSTANCE_OF  OPTIONAL NOTE	SQLCARD - SQL Communications Area Reply Data  Returned if RDBAFLRM is returned. An SQLCARD always follows the RDBAFLRM.
	NOTE	Returned following the ACCRDBRM if there is an SQL warning and/or server-specific connect tokens for a successful database connection.

	MINLVL	7
	MTLINC	X'002F' - TYPDEFNAM - Data Type Definition Name
	NOTE	If RDBAFLRM is returned.
	MTLINC	X'0035' - TYPDEFOVR - TYPDEF Overrides
	NOTE	If RDBAFLRM is returned.
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'2201'	INSTANCE_OF	ACCRDBRM - Access to RDB Completed
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported
X'1218'	INSTANCE_OF	MGRDEPRM - Manager Dependency Error
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'2207'	INSTANCE_OF	RDBACCRM - RDB Currently Accessed
X'221A'	INSTANCE_OF NOTE	RDBAFLRM - RDB Access Failed Reply Message When RDBAFLRM is returned, the target SQLAM instance is destroyed.
X'2203'	INSTANCE_OF	RDBATHRM - Not Authorized to RDB
X'2211'	INSTANCE_OF	RDBNFNRM - RDB Not Found
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limits Reached
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'125F'	INSTANCE_OF	TRGNSPRM - Target Not Supported
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

**SEE ALSO**

<b>insvar</b>	<i>ACCRDBRM</i> (on page 48) <i>PRCCNVCD</i> (on page 621)
<b>clscmd</b>	<i>SQLAM</i> (on page 847)
<b>cmddda</b>	<i>BNDSQLSTT</i> (on page 136) <i>DSCRDBTBL</i> (on page 300) <i>EXCSQLIMM</i> (on page 371) <i>EXCSQLSTT</i> (on page 381) <i>OPNQRY</i> (on page 555) <i>PRPSQLSTT</i> (on page 636)
<b>Semantic</b>	<i>ACCRDBRM</i> (on page 48) <i>BGNBND</i> (on page 110) <i>BNDSQLSTT</i> (on page 136) <i>CLSQR</i> (on page 165) <i>CNTQRY</i> (on page 222) <i>DFTPKG</i> (on page 273) <i>DRPPKG</i> (on page 293)

† This instance variable is OPTIONAL in DDM Level 3.

*DSCRDBTBL* (on page 300)  
*DSCSQLSTT* (on page 304)  
*ENDBND* (on page 336)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSET* (on page 377)  
*EXCSQLSTT* (on page 381)  
*INTRDBRQS* (on page 445)  
*OPNQRY* (on page 555)  
*PRDDTA* (on page 630)  
*PRPSQLSTT* (on page 636)  
*PWDSBS* (on page 646)  
*RDBACCRM* (on page 724)  
*RDBCMM* (on page 728)  
*RDBNACRM* (on page 734)  
*RDBOVR* (on page 740)  
*RDBRLLBCK* (on page 745)  
*REBIND* (on page 753)  
*RSCLMTRM* (on page 778)  
*SQL* (on page 835)  
*SQLAM* (on page 847)  
*SRVLST* (on page 878)  
*SYNCPTOV* (on page 944)  
*TYPDEFOVR* (on page 1030)

NAME

ACCRDBRM — Access to RDB Completed

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'2201'  
**Length** \*  
**Class** CLASS  
**Sprcls** RPYMSG - Reply Message

Access to RDB Completed (ACCRDBRM) Reply Message specifies that an instance of the SQL application manager has been created and is bound to the specified relational database (RDB).

The *crrtkn* parameter carries information to correlate with the work being done on behalf of an application at the source and at the target server.

The *pkgdfcst* parameter specifies the default SQL character subtype used for any character columns defined by an SQL CREATE or ALTER table statement not having an explicit subtype specified.

The *prdid* parameter provides the source server with the product release level of the target RDB server.

The *rdbinttkn* parameter returns an interrupt token that a target SQLAM generates which the requester uses to interrupt a DDM command. If the target SQLAM does not specify the *rdbinttkn* parameter, the requester cannot interrupt the execution of the DDM commands. If the *rdbinttkn* parameter is present, the target server may optionally provide an address at which an interrupt request should be directed. The choice of a TCP/IP or SNA address should be consistent with the communication layer that is being used for the connection. For TCP/IP, the address may be returned using the *ipaddr* or *tcpporthost* parameter. For SNA, the address must be returned using the *snaaddr* parameter.

The *svlst* parameter is returned if the RDB is managed by a group of servers. The purpose is to list the addresses that may be used by the source server for workload balancing.

The *typdefnam* parameter specifies the name of the data type to the data representation mapping definitions that the target SQLAM uses when sending reply data objects.

The *typdefovr* parameter specifies the single-byte, double-byte, and mixed-byte CCSIDs of the scalar data arrays (SDA) in the identified data type to the data representation mapping definitions. If the target SQLAM cannot support the *typdefovr* parameter values specified for the double-byte and mixed-byte CCSIDs on the corresponding ACCRDB command, then the severity code WARNING is specified on the ACCRDBRM.

The *usrld* parameter specifies the target-defined user ID.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL
<b>insvar</b>	CLASS INSTANCE VARIABLES

length	*	
class	X'2201'	
crtrkn	INSTANCE_OF OPTIONAL DFTVAL NOTE NOTE	CRRTKN - Correlation Token " Source server product-specific value is used. This parameter is returned if and only if the CRRTKN parameter is not received on ACCRDB.
ipaddr	INSTANCE_OF MTLEXC MTLEXC  OPTIONAL IGNORABLE NOTE	IPADDR - TCP/IP Address X'11E9' - SNAADDR - SNA Address X'1912' - TCPPOPTHOST - TCP/IP Port Number and Domain-qualified Host Name  If returned, it is the address to be used to send INTRDBRQS. This parameter can only be returned if RDBINTTKN is returned.
pkgdftcst	INSTANCE_OF ENUVAL ENUVAL ENUVAL OPTIONAL	PKGDFTCST - Package Default Character Subtype 'CSTBITS' 'CSTSBCS' 'CSTMBCS'
prdid	INSTANCE_OF REQUIRED	PRDID - Product-specific Identifier
rdbinttkn	INSTANCE_OF OPTIONAL IGNORABLE MINLVL NOTE	RDBINTTKN - RDB Interrupt Token  7 This parameter can only be returned if the requester is at RDB level 7.
snaaddr	INSTANCE_OF MTLEXC MTLEXC  OPTIONAL IGNORABLE NOTE	SNAADDR - SNA Address X'11E8' - IPADDR - TCP/IP Address X'1912' - TCPPOPTHOST - TCP/IP Port Number and Domain-qualified Host Name  If returned, it is the address to be used to send INTRDBRQS. This parameter can only be returned if RDBINTTKN is returned.
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
srvlst	INSTANCE_OF OPTIONAL MINLVL	SRVLST - Server List  5
svrcod	INSTANCE_OF REQUIRED	SVRCOD - Severity Code <QDDBASD>

	ENUVAL	0 - INFO - Information Only Severity Code <QDDPRMD>
	ENUVAL	4 - WARNING - Warning Severity Code
tcpporthost	INSTANCE_OF	TCPPORTHOST - TCP/IP Port Number and Domain-qualified Host Name
	MTLEXC	X'11E8' - IPADDR - TCP/IP Address
	MTLEXC	X'11E9' - SNAADDR - SNA Address
	MINLVL	7
	OPTIONAL	
	IGNORABLE	
	NOTE	If returned, it is the address to be used to send INTRDBRQS. This parameter can only be returned if RDBINTTKN is returned.
typdefnam	INSTANCE_OF	TYPDEFNAM - Data Type Definition Name
	ENUVAL	'QTDSQL370'
	ENUVAL	'QTDSQL400'
	ENUVAL	'QTDSQLX86'
	ENUVAL	'QTDSQLASC'
	MINLVL	4
	ENUVAL	'QTDSQLVAX'
	MINLVL	4
	REQUIRED	
typdefovr	INSTANCE_OF	TYPDEFOVR - TYPDEF Overrides
	REQUIRED	
usrld	INSTANCE_OF	USRID - User ID at the Target System
	OPTIONAL	
clscmd	NIL	
inscmd	NIL	

## SEE ALSO

<b>insvar</b>	<i>IPADDR</i> (on page 450) <i>MTLEXC</i> (on page 524) <i>QDDBASD</i> (on page 651) <i>QDDPRMD</i> (on page 654) <i>QDDRDBD</i> (on page 657) <i>RDBINTTKN</i> (on page 733) <i>SNAADDR</i> (on page 827)
<b>rpydta</b>	<i>BGNATMCHN</i> (on page 107) <i>BGNBND</i> (on page 110) <i>BNSQLSTT</i> (on page 136) <i>CLSQR</i> (on page 165) <i>CNTQR</i> (on page 222) <i>DRPPKG</i> (on page 293) <i>DSCRDBTBL</i> (on page 300) <i>DSCSQLSTT</i> (on page 304) <i>ENDATMCHN</i> (on page 333) <i>ENDBND</i> (on page 336) <i>EXCSQLIMM</i> (on page 371) <i>EXCSQLSTT</i> (on page 381) <i>INTRDBRQS</i> (on page 445) <i>OPNQRY</i> (on page 555)



	<i>PRPSQLSTT</i> (on page 636)
	<i>RDBCMM</i> (on page 728)
	<i>RDBRLLBCK</i> (on page 745)
	<i>REBIND</i> (on page 753)
<b>cmdrpy</b>	<i>ACCRDB</i> (on page 42)
<b>Semantic</b>	<i>ACCRDB</i> (on page 42)
	<i>INTTKNRM</i> (on page 448)
	<i>RDBINTTKN</i> (on page 733)
	<i>SRVLST</i> (on page 878)
	<i>SYNCPTOV</i> (on page 944)

**NAME**

ACCSEC — Access Security

**DESCRIPTION (Semantic)****Dictionary** QDDBASD**Codepoint** X'106D'**Length** \***Class** CLASS**Sprcls** COMMAND - Command

The Access Security (ACCSEC) command initializes the security mechanism.

**Source System Processing**

The source system determines the location of the security manager:

Local            Call the local security server.

Remote          Send the ACCSEC command to the remote security server.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**DSS Carrier: RQSDSS****Target System Processing**

The normal response to this command is:

```
ACCSECRD ( SECMEC ( value { value ... } ) )
```

from the target server. If the target server supports the SECMEC requested by the source server, then a single value is returned for SECMEC and it is identical to the SECMEC value in the ACCSEC command. For the selected security mechanism, if the encryption algorithm and encryption key length are specified by the source server, and if the target server supports the ENCALG and ENCKEYLEN requested, then a single value is returned for ENCALG and ENCKEYLEN and it is identical to the ENCALG value and ENCKEYLEN value in the ACCSEC command.

If the Kerberos security mechanism (KERSEC) is returned as a SECMEC on the ACCSECRD reply data object, it is highly recommended but nonetheless not mandatory that the server also returns its Kerberos principal in a KERSECPPL reply data object following the ACCSECRD.

If the target server does not support the SECMEC requested by the source server, then one or more values in SECMEC are returned. If the target server does not support the SECMEC requested by the source server and if ENCALG and ENCKEYLEN are specified by the source server, then only one or more values in SECMEC are returned. If the target server supports the SECMEC requested by the source server and if ENCALG and ENCKEYLEN are specified by the source server, and if the target server does not support the ENCALG/ENCKEYLEN requested, then a single value is returned for SECMEC and one or more values in ENCALG/ENCKEYLEN that the target server supports for the requested SECMEC are returned. The source server may select one of the values returned or terminate the network connection.

If the source server selects a security mechanism that requires additional ACCSEC/ACCSECRD variables to be exchanged, such as an ACCSEC/ACCSECRD exchange of SECTKN values, the source server will flow another ACCSEC with the selected SECMEC and the additional variables. The target server will accept another ACCSEC and return an ACCSECRD with the selected SECMEC and the required variables.

Once ACCSEC/ACCSECRD processing is complete, the source server will present the selected SECMEC in the SECCHK.

If the plug-in security mechanism (PLGIN) is returned as a SECMEC on the ACCSECRD reply data object, it must be accompanied by the list of supported plug-in modules in the PLGINLST instance variable following the ACCSECRD.

### **Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

### **Exceptions**

If the ACCSEC command is chained to the EXCSAT command and the target server does not support the SECMGR at DDM manager Level 5 or higher, then the ACCSEC is rejected with the CMDNSPRM.

### **Source System Reply Processing**

If the target security manager returned the same security mechanism combination, then use the requested security mechanism combination.

If the target security manager returned a list of security mechanism combinations, then select one of the security mechanism combinations to use. If the target server returned the same security mechanism combination and a list of encryption algorithm and encryption key length combinations, then select one of the encryption algorithm and encryption key length combinations to use. If the source server does not support any of the combinations returned in the list or none are acceptable to the source server, then the source server must terminate the network connection and return a security error to the user.

If the source server does support one or more of the security mechanisms in the list, and for the selected security mechanism, if the source server does support the required encryption algorithm and required encryption key length, then it continues. When continuing, if the selected security mechanism requires that additional ACCSEC/ACCSECRD variables be exchanged, such as an ACCSEC/ACCSECRD exchange of SECTKN values, the source server must flow another ACCSEC with the selected SECMEC and the additional variables. The target server will accept another ACCSEC and return an ACCSECRD with the selected SECMEC and the required variables.

Once ACCSEC/ACCSECRD processing is complete, the source server should continue processing with a SECCHK command containing the selected SECMEC.

Return any reply messages and data objects to the requester.

### Handling Subsequent ACCSEC Commands

The source server can send repeated ACCSEC commands when it wants to reuse a connection for another application or transaction. The ACCSEC must be preceded by an EXCSAT, and followed by a SECCHK and ACCRDB.

### Data Conversion

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

clsvar	NIL	
insvar	CLASS INSTANCE VARIABLES	
length	*	
class	X'106D'	
encalg	INSTANCE_OF OPTIONAL NOTE	ENCALG - Encryption Algorithm  This parameter applies if the security mechanism is EUSRIDPWD, EUSRIDNWPWD, EUSRIDDTA, EUSRPWDDTA, or EUSRNPWDDTA. Otherwise, this parameter ignored.
	MINLVL	7
enckeylen	INSTANCE_OF OPTIONAL NOTE	ENCKEYLEN - Encryption Key Length  This parameter applies if the security mechanism is EUSRIDPWD, EUSRIDNWPWD, EUSRIDDTA, EUSRPWDDTA, or EUSRNPWDDTA. Otherwise, this parameter is ignored.
	MINLVL	7
plginid	INSTANCE_OF OPTIONAL NOTE	PLGINID - Security Plug-in-specific Identifier  This parameter may optionally be used to convey the plug-in-specific version information if PLGINNM is specified. Otherwise, this parameter is not used.
	MINLVL	7
plginnm	INSTANCE_OF OPTIONAL NOTE	PLGINNM - Security Plug-in Name  This parameter may optionally be used to indicate a preferred plug-in module if the security mechanism is PLGIN. Otherwise, this parameter is not used.
	MINLVL	7
rdbnam	INSTANCE_OF	RDBNAM - Relational Database Name

	OPTIONAL NOTE	This parameter may be required if the target server owns multiple RDBs, and allows each RDB to have its own security mechanism.
secmec	INSTANCE_OF REQUIRED	SECMEC - Security Mechanism
secmgrnm	INSTANCE_OF OPTIONAL IGNORABLE NOTE	SECMGRNM - Security Manager Name  This parameter has a null value and does not have to be validated.
	CMDTRG	
sectkn	INSTANCE_OF OPTIONAL NOTE	SECTKN - Security Token  This parameter is required if the security mechanism is USRSBSPWD, USRSSBPWD, USRENCPWD, EUSRIDPWD, or EUSRIDNWPWD. See USRSECOVR for details.
	MINLVL NOTE	6 This parameter is required if the security mechanism is EUSRIDDTA, EUSRPWDDTA, or EUSRNPWDDTA. See EDTASECOVR for details.
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>cmddta</b>	<b>NIL</b>	
<b>rpydta</b>		<b>REPLY OBJECTS</b>
X'14AC'	INSTANCE_OF REQUIRED NOTE	ACCSECRD - Access Security Reply Data  If the ACCSEC command fails, then the ACCSECRD is not sent and instead one of the reply messages is sent.
X'1C02'	INSTANCE_OF OPTIONAL NOTE	KERSECPPL - Kerberos Security Principal  If the Kerberos security mechanism (KERSEC) is returned by the target server as a supported security mechanism on the ACCSECRD, it is highly recommended that its Kerberos principal be returned in the KERSECPPL.
	MINLVL	7
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported

X'123C'	INSTANCE_OF	INVRQSRM - Invalid Request
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'221A'	INSTANCE_OF NOTE	RDBAFLRM - RDB Access Failed Reply Message Only provided if RDBNAM is provided. When RDBAFLRM is returned, the target SQLAM instance is destroyed.
X'2211'	INSTANCE_OF NOTE	RDBNFNRM - RDB Not Found If the specified <i>rdbnam</i> is not found, or if the target server requires the presence of an <i>rdbnam</i> but the <i>rdbnam</i> was not provided, RDB Not Found is returned.
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limits Reached
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'125F'	INSTANCE_OF	TRGNSPRM - Target Not Supported
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

**SEE ALSO**

<b>inscmd</b>	<i>SECMGR</i> (on page 814)
<b>insvar</b>	<i>ACCSECRD</i> (on page 58) <i>PRCCNVCD</i> (on page 621) <i>SECCHKCD</i> (on page 806)
<b>Semantic</b>	<i>ACCSECRD</i> (on page 58) <i>DCESEC</i> (on page 252) <i>DCESECOVR</i> (on page 253) <i>DHENC</i> (on page 280) <i>ENCALG</i> (on page 331) <i>ENCKEYLEN</i> (on page 332) <i>EDTASECOVR</i> (on page 323) <i>EUSRIDDTA</i> (on page 355) <i>EUSRIDNWPWD</i> (on page 357) <i>EUSRIDPWD</i> (on page 358) <i>EUSRNPWDDTA</i> (on page 359) <i>EUSRPWDDTA</i> (on page 361) <i>KERSECOVR</i> (on page 466) <i>MGRLVLN</i> (on page 509) <i>PLGIN</i> (on page 609) <i>PLGINOVR</i> (on page 615) <i>PWDENC</i> (on page 644) <i>PWDSBS</i> (on page 646) <i>SECCHK</i> (on page 800) <i>SECMGR</i> (on page 814) <i>SECTKN</i> (on page 820) <i>SYNCPTOV</i> (on page 944) <i>TCPCMNI</i> (on page 994) <i>USRENCPWD</i> (on page 1043) <i>USRIDNWPWD</i> (on page 1045) <i>USRSBSPWD</i> (on page 1049) <i>USRIDONL</i> (on page 1046)

*USRIDPWD* (on page 1047)  
*USRSECOVR* (on page 1050)

NAME

ACCSECRD — Access Security Reply Data

DESCRIPTION (Semantic)

**Dictionary** QDDBASD  
**Codepoint** X'14AC'  
**Length** \*  
**Class** CLASS  
**Sprcls** COLLECTION - Collection Object

The Access Security Reply Data (ACCSECRD) Collection Object contains the security information from a target server's security manager. This information is returned in response to an ACCSEC command.

If an error is detected in processing the ACCSEC command, a SECCHKCD is returned with the ACCSECRD. It has an implied severity code of ERROR. The ACCSEC command must be sent again to generate a new set of instance variables on the ACCSECRD prior to sending the SECCHK command to authenticate the connection.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'14AC'	
<b>encalg</b>	INSTANCE_OF OPTIONAL NOTE	ENCALG - Encryption Algorithm  This parameter applies if the security mechanism is EUSRIDPWD, EUSRIDNWPWD, EUSRIDDTA, EUSRPWDDTA, or EUSRNPWDDTA. Otherwise, this parameter is ignored.
	NOTE	The ENCALG parameter must either reflect the value sent in the ACCSEC command if the target server supports it; or the values the target server does support for the requested SECMEC, when it does not support or accept the value requested by the source server.
	MINLVL	7
<b>enckeylen</b>	INSTANCE_OF OPTIONAL NOTE	ENCKEYLEN - Encryption Key Length  This parameter applies if the security mechanism is EUSRIDPWD, EUSRIDNWPWD, EUSRIDDTA, EUSRPWDDTA, or EUSRNPWDDTA. Otherwise, this parameter is ignored.
	NOTE	The ENCKEYLEN parameter must either reflect the value sent in the ACCSEC command if the target server supports it; or the values the target server does support for the requested SECMEC,



	MINLVL	7	when it does not support or accept the value requested by the source server.
plginlst	INSTANCE_OF OPTIONAL NOTE	PLGINLST - Security Plug-in List	This parameter is used to provide a list of plug-in module names supported at the target server if the security mechanism is PLGIN. Otherwise, this parameter is not used. The list entries should be ordered from highest to lowest preference. If the PLGINNM provided in the ACCSEC command is accepted, then it will be returned as the sole entry in the plug-in list.
secchkcd	MINLVL INSTANCE_OF OPTIONAL NOTE	SECCHKCD - Security Check Code	This parameter is included if and only if an error is detected when processing the ACCSEC command. The implied severity code is ERROR.
secmec	INSTANCE_OF NOTE	SECMEC - Security Mechanism	The SECMEC parameter must either reflect the value sent in the ACCSEC command if the target server supports it; or the values the target server does support when it does not support or accept the value requested by the source server.
sectkn	REQUIRED INSTANCE_OF OPTIONAL NOTE	SECTKN - Security Token	This parameter is required if the security mechanism is USRSBSPWD, USRSSBPWD, USRENCPWD, EUSRIDPWD, or EUSRIDNWPWD. See USRSECOVR for details.
	MINLVL NOTE	6	This parameter is required if the security mechanism is EUSRIDDTA, EUSRPWDDTA, or EUSRNPWDDTA. See EDTASECOVR for details.
	MINLVL	7	
clscmd	NIL		
inscmd	NIL		

**SEE ALSO**

<b>insvar</b>	<i>PRCCNVCD</i> (on page 621) <i>SECMEC</i> (on page 811)
<b>rpydta</b>	<i>ACCSEC</i> (on page 52)
<b>Semantic</b>	<i>ACCSEC</i> (on page 52) <i>ACCSECRD</i> (on page 58) <i>DCESECOVR</i> (on page 253) <i>DHENC</i> (on page 280)

*EDTASECOVR* (on page 323)  
*ENCALG* (on page 331)  
*ENCKEYLEN* (on page 332)  
*EUSRIDDTA* (on page 355)  
*EUSRIDNWPWD* (on page 357)  
*EUSRIDPWD* (on page 358)  
*EUSRNPWDDTA* (on page 359)  
*KERSECOVR* (on page 466)  
*PLGIN* (on page 609)  
*PLGINOVR* (on page 615)  
*PWDSBS* (on page 646)  
*SYNCPTOV* (on page 944)  
*TCPCMNI* (on page 994)  
*USRSBSPWD* (on page 1049)  
*USRSECOVR* (on page 1050)

**NAME**

AGENT — Agent

**DESCRIPTION (Semantic)****Dictionary** QDDBASD**Codepoint** X'1403'**Length** \***Class** CLASS**Sprcls** MANAGER - Resource Manager

Agent (AGENT) Resource Manager is one of the basic operative parts of the DDM architecture. An Agent represents a requester (a file-requesting routine) to a server. This representation includes the following:

- Interfacing with the requester to receive requests and pass back responses in the same order as the received requests.
- Interfacing to its local server to determine where the command is sent to be processed, to locate and allocate resources, and to enforce security.

The agent represents the requester to the local server's supervisor which controls and accounts for the memory, processor, file storage, spooling, and other resources a user needs.

An agent represents the requester to its server's security manager. The security manager validates each request for a resource and each command to a file.

- Processing commands related to agent functions and returning the results to the requester.
- Interfacing to the appropriate communications manager for any communications that are required.

A source agent can manage communications with one or more target agents on behalf of the same requester. A single communications connection with a single target agent can access multiple files controlled by the same target server.

- If the agent needs to use the SECMGR at DDM manager Level 5, then the agent can be at any manager level. See *SECMGR* (on page 814) for the functions of the SECMGR at manager Level 5.
- Interfacing to SQL application managers for any accessed relational database managers.
- Modifying the operational processes to use alternate logic, including:
  - Modifying the command and reply message parameter syntax as an ordered collection following the order in the DDM architecture instead of the unordered collection.
  - Modifying the reply message syntax to restrict the reply message to just the reply message codepoint instead of the entire reply message.
  - Modifying the communications protocol to optionally hold reply messages rather than always sending them when required.
  - Modifying the handling of large objects to add prefix and suffix objects (bookends) around them rather than specifying the true length of the large object.
- The agent can use the CCSID to transform the variables in the DDM commands and reply messages that contain character data from one code page to another (see *CCSIDMGR* (on page 150)). This allows products to support the system code page especially when the

system code page is not CCSID 500. For example, two products with system code page ASCII 819 that support the CCSIDMGR can send character data in the DDM commands and reply messages in that code page and not need to transform the character data into EBCDIC. Additionally, two products with different system code pages can send character data in the local system code page. The receiver of the DDM commands and reply messages must transform the character data to its system code page.

- The agent at Level 5 supports the processing of a new type of request (DSS type X'5') that does not require a reply to be returned from the target server. Interfacing to SYNCPTMGR Level 5 requires the support for the new DSS type. The source SYNCPTMGR sends a SYNCCTL command where no response is expected from the target SYNCPTMGR when the command is processed successfully.
- The agent at Level 7 supports the processing of the SNDPKT command to test its connectivity from source server to target server.

### Agent Processing

The following terms illustrate the processing the source and target agents perform for a variety of conditions:

- AGNCMDPR      Target Agent Command Processing (see *AGNCMDPR* (on page 64))
- AGNDCLPR      Target Agent Declare Name Processing
- AGNRPYPR      Source Agent Reply Message Processing (see *AGNRPYPR* (on page 67))

### Manager-Level Compatibility

Table 3-1 illustrates the function of the AGENT as it has grown and changed through the levels of the DDM architecture.

**Table 3-1** Agent-Level Compatibility

<b>DDM Levels</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>
<i>Manager Dependencies</i>							
AGENT	1	2	3	4	5	6	7
DICTIONARY	1	2	3	4	5	6	7
RDB			3	3	3	3	7
CCSIDMGR				4	4	4	4
<i>Manager Routed-to</i>							
SUPERVISOR	1	1	1	1	1	1	1
SECMGR†	1	1	1	1	5	6	7
SQLAM			3	4	5	6	7
RSYNCMGR					4	4	4
SYNCPTMGR					5	5	7
XAMGR							7
<i>Communication Managers</i>							
CMNAPPC	1	1	3	3	3	3	3
CMNSYNCPT				4	4	4	4
CMNTCPIP					5	5	5

<b>mgrlvl</b>	5	
<b>mgrdepls</b>		<b>MANAGER DEPENDENCY LIST</b>
X'1458'	INSTANCE_OF NOTE	DICTIONARY - Dictionary Dictionaries resolve codepoints to class descriptors for validation and routing functions.
	MGRVLN	
X'1440'	INSTANCE_OF NOTE	SECMGR - Security Manager The Security Manager validates the authorizations of the requester to specific commands and dictionary classes.
	MGRVLN	5
	NOTE	The SECMGR can be at DDM Level 1 or DDM Level 5.
	MINLVL	5
X'14CC'	INSTANCE_OF NOTE	CCSIDMGR - CCSID Manager The CCSIDMGR transforms received character data in the DDM commands and reply messages that were sent in a different code page than the agent and its local system is using for character data.
	MGRVLN	4
	MINLVL	4
	OPTIONAL	
<b>vldattls</b>		<b>VALID ATTRIBUTES</b>
X'1152'	INSTANCE_OF	AGNNAM - Agent Name
X'0019'	INSTANCE_OF	HELP - Help Text
X'0045'	INSTANCE_OF	TITLE - Title

**SEE ALSO**

<b>mgrdepls</b>	<i>RSYNMGR</i> (on page 787) <i>SQLAM</i> (on page 847) <i>SYNCPTMGR</i> (on page 939) <i>XAMGR</i> (on page 1063)
<b>Semantic</b>	<i>EXTENSIONS</i> (on page 400) <i>INHERITANCE</i> (on page 437) <i>MANAGER</i> (on page 491) <i>RDBOVR</i> (on page 740) <i>RSYNMGR</i> (on page 787) <i>SQLAM</i> (on page 847) <i>SUBSETS</i> (on page 902) <i>SYNCPTMGR</i> (on page 939)

---

† The SECMGR can be at DDM Level 1 or DDM Level 5.

**NAME**

AGNCMDPR — Target Agent Command Processing

**DESCRIPTION (Semantic)**

**Dictionary** QDDTTRD

**Length** \*

**Class** HELP

Target Agent Command Processing (AGNCMDPR) describes how the target agent processes commands.

When a target communications manager (TCM) receives a chain of RQSDSSs and OBJDSSs from a source communications manager, the TCM:

1. Extracts each command from its RQSDSS and associates the request correlation identifier of the RQSDSS with the command.
2. Extracts all of the objects from each OBJDSS and associates the request correlation identifier of the OBJDSS with each object.
3. Passes commands one at a time to the target agent with its request correlation identifier.
4. Passes data objects one at a time to the target agent as the request correlation identifier requests them.

**SEE ALSO**

**Semantic**        *AGENT* (on page 61)  
                  *CMDTRG* (on page 179)  
                  *COMMAND* (on page 240)  
                  *SUBSETS* (on page 902)

**NAME**

AGNPRMRM — Permanent Agent Error

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1232'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

Permanent Agent Error (AGNPRMRM) Reply Message indicates that the command requested could not be completed because of a permanent error condition detected at the target system.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'1232'	
<b>svrcod</b>	INSTANCE_OF REQUIRED	SVRCOD - Severity Code
	ENUVAL	16 - SEVERE - Severe Error Severity Code
	ENUVAL	32 - ACCDMG - Access Damage Severity Code
	ENUVAL	64 - PRMDMG - Permanent Damage Severity Code
<b>recnt</b>	INSTANCE_OF MINVAL OPTIONAL NOTE	RECCNT - Record Count 0 Required for requests to insert multiple records in a file. This parameter is not returned by commands that operate on RDBs.
	MINLVL	
<b>rdbnam</b>	INSTANCE_OF MINLVL OPTIONAL	RDBNAM - Relational Database Name 3
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

- cmdrpy** ACCRDB (on page 42)
- ACCSEC (on page 52)
- BGNATMCHN (on page 107)
- BGNBND (on page 110)
- BNDSQLSTT (on page 136)
- CNTQRY (on page 222)
- DRPPKG (on page 293)
- DSCRDBTBL (on page 300)

	<i>DSCSQLSTT</i> (on page 304)
	<i>ENDATMCHN</i> (on page 333)
	<i>ENDBND</i> (on page 336)
	<i>EXCSAT</i> (on page 363)
	<i>EXCSQLIMM</i> (on page 371)
	<i>EXCSQLSET</i> (on page 377)
	<i>INTRDBRQS</i> (on page 445)
	<i>OPNQRY</i> (on page 555)
	<i>PRPSQLSTT</i> (on page 636)
	<i>RDBCMM</i> (on page 728)
	<i>RDBRLLBCK</i> (on page 745)
	<i>REBIND</i> (on page 753)
	<i>SECCHK</i> (on page 800)
	<i>SYNCCTL</i> (on page 915)
	<i>SYNCRSY</i> (on page 982)
<b>rpydta</b>	<i>COMMAND</i> (on page 240)
<b>Semantic</b>	<i>DSS</i> (on page 308)
	<i>SYNCPTOV</i> (on page 944)
	<i>TCPTRGER</i> (on page 1015)



**NAME**

AGNRPYPR — Source Agent Reply Message Processing

**DESCRIPTION (Semantic)**

**Dictionary** QDDTTRD

**Length** \*

**Class** HELP

Source Agent Reply Message Processing (AGNRPYPR) describes how the source communications manager (SCM) processes a reply message which it receives from the target communications manager (TCM).

When a source communications manager receives RPYDSSs from a target communications manager, it extracts the reply messages from the RPYDSSs. The SCM then associates the reply messages with the request correlation (RQSCRR) of the RPYDSS. As each reply message is extracted, it is passed to the source agent that originated the request.

**Agent Processing Options**

The process and the options for each process are as follows:

1. Parameter Sequence

The parameters or instance variables of the DDM commands and reply messages are arranged in the following manner:<sup>1</sup>

**ORDERED** The parameters must be sent in the order that the DDM Architecture defines.

2. Large Objects

**EXTLL** This is the original processing method. The high-order bit in the two-byte length is set to one, and an extended length (true length of the object) is added following the codepoint of the object and its envelopes.

**SEE ALSO**

**Semantic** *AGENT* (on page 61)

---

1. New parameters can be added only at the end of the list of variables in the commands and reply messages.

**NAME**

APPCMNFL — LU 6.2 Communications Failure

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

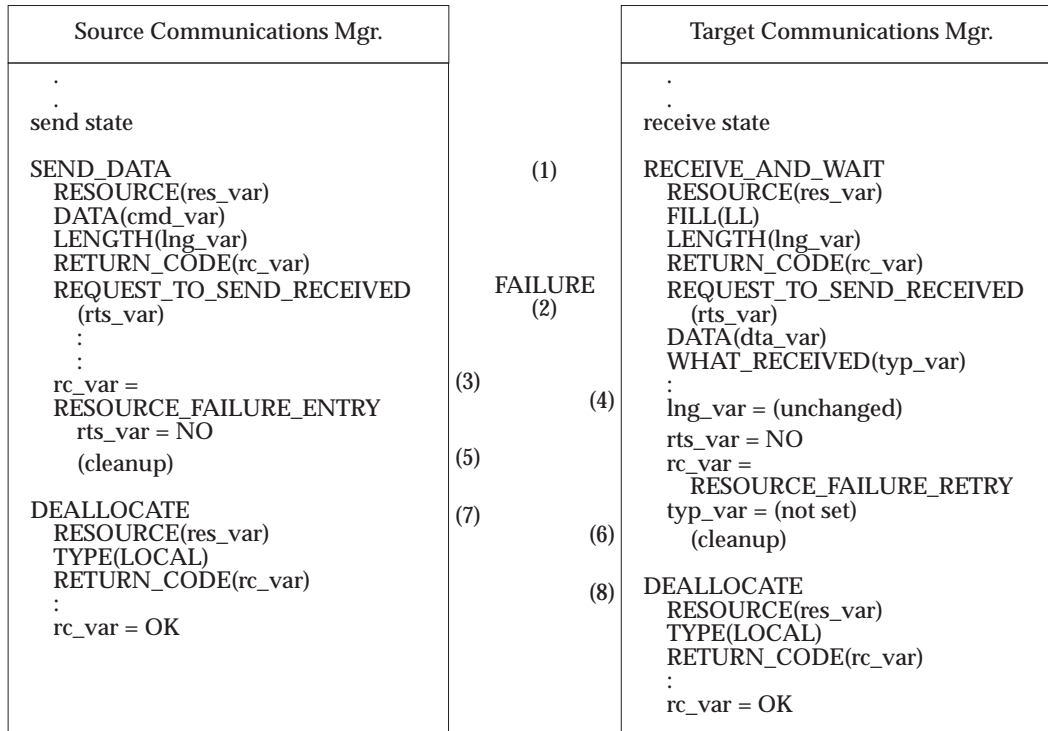
LU 6.2 Communications Failure (APPCMNFL) illustrates the communications sequence occurring when a communications failure prematurely terminates the communications between the source communications manager (SCM) and the target communications manager (TCM). An SNA LU 6.2 session protocol error, an SNA LU 6.2 session outage, a communications line failure, a modem failure, a remote system failure, or failures of many other types can result in a communication breakdown between the SCM and TCM. Do not confuse communication failures with DDM-detected errors that result in a reply message. See the SNA (System Network Architecture) manuals for detailed information about SNA communications failures.

The basic sequence for handling a communications failure is for both the SCM and the TCM to deallocate their SNA conversation and perform any required cleanup. See Figure 3-1 (on page 69).

A sample protocol sequence is shown below. This sequence is only a sample. SNA LU 6.2 functions provide so many functional capabilities that it is impossible to show all the possible sequences. Notes describing key points in the sequence follow the sequence diagram.

The following assumptions have been made for the sample protocol sequence:

- A conversation has been successfully established between the SCM and the TCM (see *APPCMNI* (on page 72)).
- The SCM and the TCM both perform synchronous sends and receives.



**Figure 3-1** Communications Failure Between Source and Target

**Figure Notes**

1. The SCM and TCM are passing RQSDSS and RPYDSS structures to each other. The SCM has issued a SEND\_DATA verb to send a RQSDSS to the TCM. The TCM has issued a RECEIVE\_AND\_WAIT verb to receive the next RQSDSS from the SCM.
2. A communications failure occurs. The communications resource (line) is broken, and the source and target systems cannot communicate.
3. The SEND\_DATA verb completes its operation on the source system with a return code that indicates that the communications resource (line) has failed. This tells the SCM that communications with the TCM is no longer possible with the current SNA LU 6.2 conversation that is allocated.

The SNA LU 6.2 communication facility sets the RETURN\_CODE parameter to RESOURCE\_FAILURE\_RETRY and the REQUEST\_TO\_SEND\_RECEIVED parameter to NO. The RESOURCE\_FAILURE\_RETRY indicates that the communications resource failed in such a way that attempts to reestablish communications may be successful.

4. The RECEIVE\_AND\_WAIT verb completes its operation on the target system with a return code indicating that the communications resource (line) has failed. This tells the TCM that communications with the SCM is no longer possible with the current SNA LU 6.2 conversation.

The SNA LU 6.2 communication facility does not change the LENGTH parameter; it sets the REQUEST\_TO\_SEND\_RECEIVED parameter to NO and the RETURN\_CODE parameter to RESOURCE\_FAILURE\_RETRY. The RESOURCE\_FAILURE\_RETRY indicates that the communications resource failed in such a way that attempts to reestablish communications

may be successful.

5. The SCM notifies the source agent and performs any required cleanup functions. This may include cleaning up internal tables and control blocks, logging the failure, and so on.
6. The TCM performs any required cleanup functions. These include:
  - Notifying the target agent of the failure. Upon notification of the failure, the target agent notifies the other server managers to perform cleanup functions. For servers that support files, these cleanup functions include:
    - Completing current DDM command processing, if possible
    - Releasing all record and stream locks that are being held
    - Closing any files that are open
    - Releasing all file locks that are being held
    - Performing any required additional cleanup, such as freeing up DCLNAMs in the DCLFIL collection
  - For servers that support relational databases, these include:
    - Performing a rollback on any accessed RDBs
    - Destroying any target SQLAM manager instances
    - Performing any additional cleanup required
  - Performing any required additional cleanup, such as cleaning up internal tables or control blocks, logging the failure, and so on

7. The SCM issues a DEALLOCATE verb to deallocate the SNA LU 6.2 conversation.

The RESOURCE parameter specifies the SNA LU 6.2 conversation resource. The TYPE parameter specifies LOCAL because no communications with the target system are available to perform the deallocate function.

The DEALLOCATE verb completes with a RETURN\_CODE of OK. The SCM is now completely disassociated from the SNA LU 6.2 communications facility on the source system.

8. The TCM issues a DEALLOCATE verb to deallocate the SNA LU 6.2 communications conversation.

The RESOURCE parameter specifies the SNA LU 6.2 conversation resource. The TYPE parameter specifies LOCAL because no communications with the source system are available to perform the deallocate function.

The DEALLOCATE verb completes with a RETURN\_CODE of OK. The TCM is now completely disassociated from the SNA LU 6.2 communications facility on the target system.

**SEE ALSO**

**Semantic**

*APPCMNT* (on page 76)

*CMNAPPC* (on page 184)

**NAME**

APPCMNI — LU 6.2 Communications Initiation

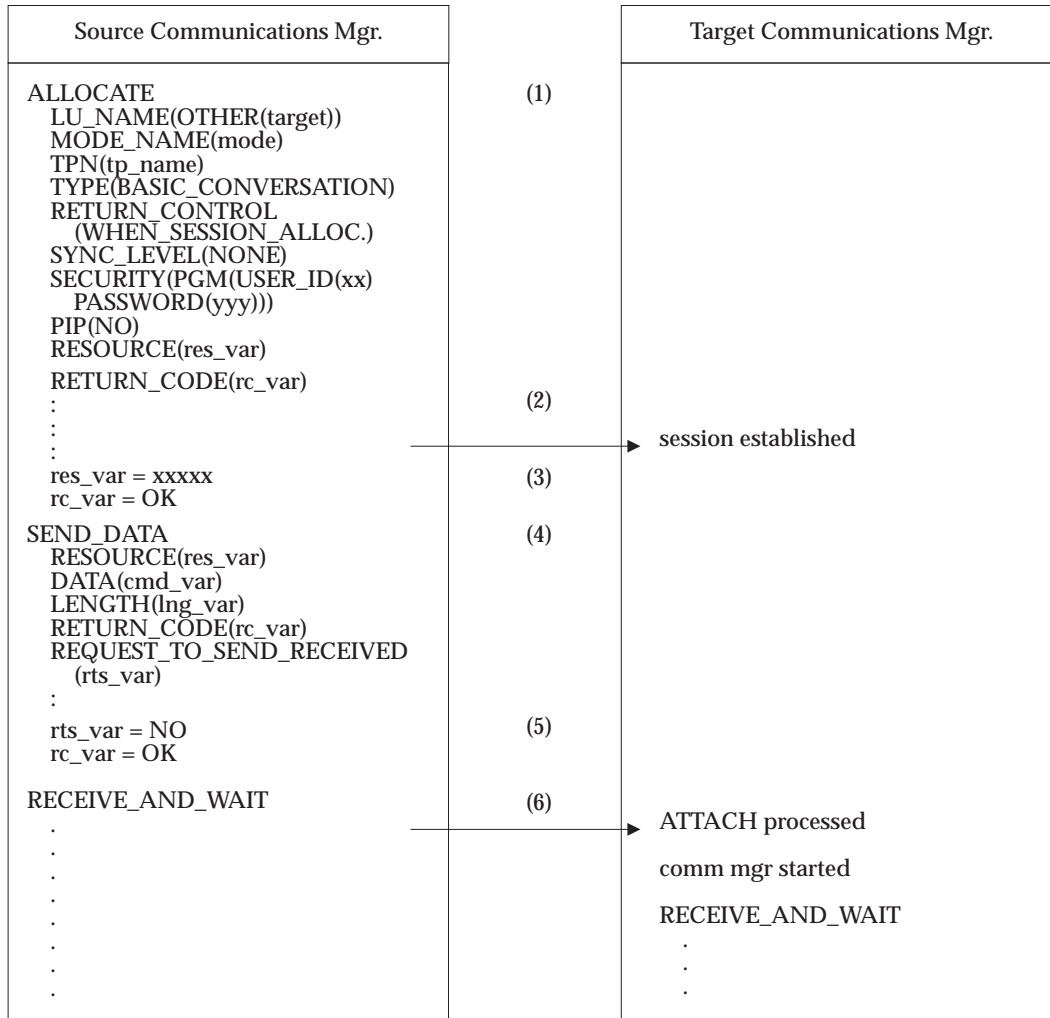
**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

LU 6.2 Communications Initiation (APPCMNI) illustrates the use of SNA LU 6.2 communications facilities to initiate source-to-target communications. The SNA LU 6.2 communications facility is responsible for LU 6.2 session initiation. More information about SNA LU 6.2 session initiation is in the SNA LU 6.2 manuals.

A sample protocol sequence is shown in Figure 3-2 (on page 73). SNA LU 6.2 functions provide such an extensive set of functional capabilities that it is impossible to show all the possible sequences. Notes describing key points in the sequence follow the sequence diagram in Figure 3-2 (on page 73).

The following assumptions have been made for the sample protocol sequence:

- No conversation exists between the source communications manager (SCM) and the target communications manager (TCM).
- Both the SCM and the TCM can perform synchronous sends and receives.
- The session limit between the source and target LUs has not been reached.
- No error situations occur.



**Figure 3-2** Communications Initiation by Source System

**Figure Notes**

1. The SCM establishes communications with the TCM by issuing an ALLOCATE verb to the SNA LU 6.2 communication facility. The ALLOCATE verb contains the information the SNA LU 6.2 needs to determine where the TCM is located (LU\_NAME), the type of SNA LU 6.2 services being used (MODE\_NAME), the type of verb interface the SCM uses with SNA LU 6.2 (TYPE), and the synchronization level that is used (SYNC\_LEVEL). In addition to these communications parameters, the ALLOCATE verb contains the identification of the transaction program being initiated (TPN), parameter data that the transaction program (PIP) might require, and the security information for verifying the user (SECURITY).

The SECURITY parameter specifies NONE, SAME, or PGM. If NONE is specified, some target servers might reject the allocation if they require user identification before allowing access to their resources. DDM allows the use of *already verified* APPC security functions but does not require that a target server instance support them. In this illustration, the USER\_ID and PASSWORD of the requester are sent to the target system for verification.

The ALLOCATE verb only requests that a conversation and session to the remote location (LU\_NAME) be assigned to the SCM.

2. The SNA LU 6.2 facility tries to find an existing, unused session with the remote location. If there are none, SNA LU 6.2 attempts to establish one. More information on this process is in the SNA LU 6.2 manuals.
3. When the SNA LU 6.2 communication facility has successfully assigned a conversation and session to the SCM, the RETURN\_CODE variable is set to OK. The RESOURCE variable is also set to the resource identifier for the SNA LU 6.2 conversation the SCM can use to communicate during the session. Control is returned to the SCM (because the RETURN\_CONTROL parameter specified WHEN\_SESSION\_ALLOCATED). Note that the source communications manager has not yet actually communicated with the TCM.
4. The SCM issues a SEND\_DATA verb to the SNA LU 6.2 facility which sends the first DDM command (RQSDSS) to the TCM. This first command must be an EXCSAT to determine the server and manager levels of the target server.

The RESOURCE parameter identifies the conversation resource to be used. This is the same value as the value returned on the ALLOCATE verb. The DATA parameter specifies the location of the data (RQSDSS) to be sent. The LENGTH parameter gives the total length of the data at the DATA location.

5. The data is accepted by the SNA LU 6.2 facility and queued for output. Note that it is not sent to the TCM at this time.

Once the SNA LU 6.2 facility successfully possesses the data, the RETURN\_CODE parameter is set to OK, and control is returned to the SCM process. In this example, REQUEST\_TO\_SEND\_RECEIVED is set to NO.

6. The SCM issues a RECEIVE\_AND\_WAIT verb to receive the RPYDSS or OBJDSS from the RQSDSS it just sent. This verb causes the SNA LU 6.2 facility to transmit all of the data in its transmit buffers and to send the TCM the *right to send*.

The contents of the transmission would be the following:

- A bracket bid, chaining information, and a change direction indication
- An ATTACH header carrying the transaction program name (the TCM transaction program)

The transaction program name can be any valid transaction program name that invokes a DDM communications manager (see *CMNAPPC* (on page 184)).

The ATTACH header also contains the user security information.

- Data—This is the RQSDSS the SCM built.

At this point, communications have been established with the target system. The TCM process has been initiated, and DDM commands have begun to flow.



**SEE ALSO**

**Semantic**

- APPCMNFL* (on page 68)
- APPCMNT* (on page 76)
- APPSRCCD* (on page 79)
- APPSRCCR* (on page 86)
- APPSRCER* (on page 91)
- APPTRGER* (on page 95)
- CMNAPPC* (on page 184)

**NAME**

APPCMNT — LU 6.2 Communications Termination

**DESCRIPTION (Semantic)**

**Dictionary** QDDTTRD

**Length** \*

**Class** HELP

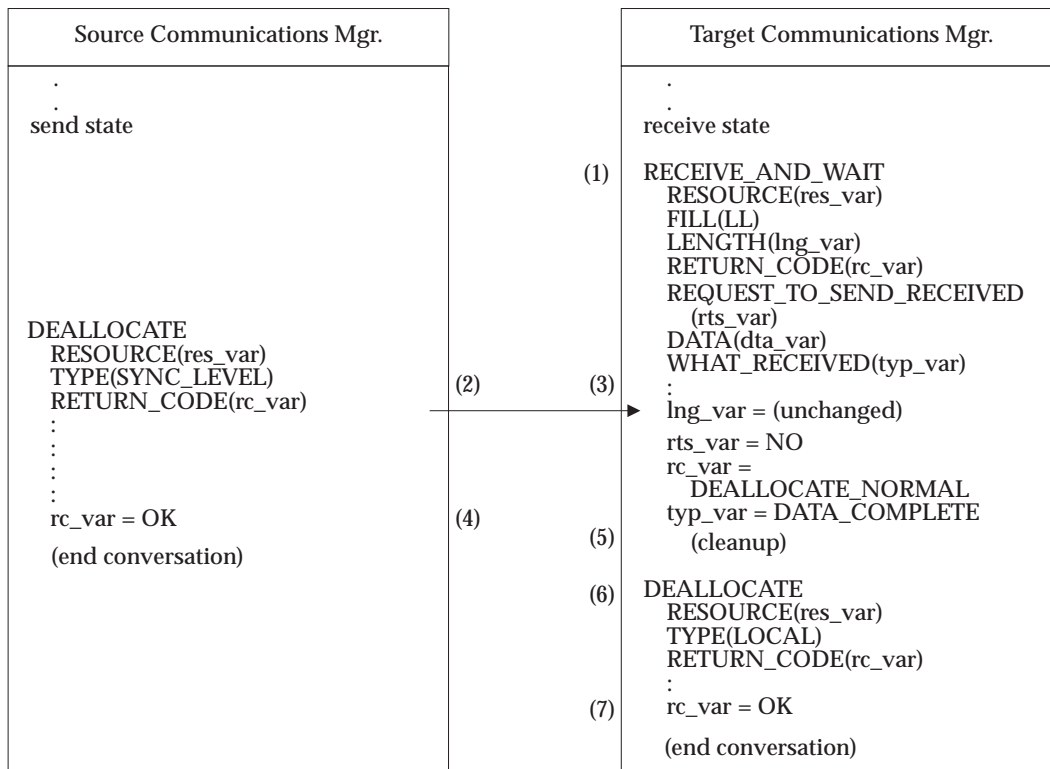
LU 6.2 Communications Termination (APPCMNT) illustrates normal communication termination by the source communications manager (SCM).

Under normal circumstances, only the SCM can terminate the conversation between the SCM and the target communications manager (TCM). See *APPCMNTFL* (on page 68) for abnormal circumstances. The SCM should terminate the conversation only when the source system has completed all of its work with the target system.

A sample protocol sequence is shown. SNA LU 6.2 functions provide such an extensive set of functional capabilities that it is impossible to show all the possible sequences. Notes describing key points in the sequence follow the sequence diagram.

The following assumptions have been made for the sample protocol sequence:

- A conversation is successfully established between the SCM and the TCM (see *APPCMNI* (on page 72)).
- No error situation occurs.



**Figure 3-3** Normal Communications Termination

**Figure Notes**

1. The TCM has issued a RECEIVE\_AND\_WAIT verb to receive the next RQSDSS from the SCM.

The RESOURCE parameter identifies the conversation resource to be used. The FILL parameter specifies that the TCM wants to receive a single logical record. The LENGTH parameter specifies the total length of the buffer space available at the DATA parameter location. Note that the buffer space should be large enough to receive the largest anticipated RQSDSS from the SCM.

2. The source system finishes using the remote data facilities of the target system. The SCM then issues a DEALLOCATE verb to deallocate the SNA LU 6.2 conversation.

The RESOURCE parameter identifies the SNA LU 6.2 conversation being deallocated, and the TYPE parameter specifies that the deallocation process uses the SYNC\_LEVEL of the conversation.

The source SNA LU 6.2 facility transmits available queued data along with the deallocation request to the TCM. This example assumes that no queued up data is available to be sent to the TCM.

The transmission contents would be:

- Chaining information and an end bracket (deallocate/detach) indication
- An LUSTAT command with a sense code of 0006

The LUSTAT is simply a carrier for the information listed above.

3. The target system receives the deallocate/detach indications and causes the RECEIVE\_AND\_WAIT verb to complete its operation. Control is returned to the TCM.

The SNA LU 6.2 communication facility does not set the LENGTH parameter. It sets the REQUEST\_TO\_SEND\_RECEIVED parameter to NO, the RETURN\_CODE parameter to DEALLOCATE\_NORMAL, and the WHAT\_RECEIVED parameter to DATA\_COMPLETE.

4. The DEALLOCATE verb completes its operation with a RETURN\_CODE of OK. The SCM is now completely dissociated from the SNA LU 6.2 communications facility on the source system.
5. When the TCM sees the DEALLOCATE\_NORMAL return code and this is the last conversation, the TCM notifies the target agent which notifies the other server's managers to perform any required cleanup.
6. The TCM deallocates its SNA LU 6.2 communication conversation.  
The RESOURCE parameter specifies the SNA LU 6.2 conversation resource. The TYPE parameter specifies LOCAL because there are no communications with the source system to perform the deallocate function.
7. The DEALLOCATE verb completes its operation with a RETURN\_CODE of OK. The TCM is now completely dissociated from the SNA LU 6.2 communications facility on the target system.

The SNA LU 6.2 session might or might not be terminated as a result of this sequence. The SNA LU 6.2 facility, not the DDM communications manager, directly controls termination.

**SEE ALSO**

**Semantic**

*CMNAPPC* (on page 184)

**NAME**

APPSRCCD — LU 6.2 Source Command with Data

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

LU 6.2 Source Command with Data (APPSRCCD) illustrates the communications sequence occurring when the source communications manager (SCM) sends an RQSDSS chain (consisting of one RQSDSS and one OBJDSS) to the target system, and the target communications manager (TCM) returns a single RPYDSS or OBJDSS.

This is the normal protocol that results when a source agent sends a DDM command followed by command data. For example, the DDM command could be an INSRECxx command followed by an OBJDSS that contains the command data (record) inserted into the file. It could be a MODREC command followed by an OBJDSS that contains a modified record, or it could be an EXCSQLIMM command followed by an OBJDSS that contains the SQL statement being executed.

The basic sequence for the SCM is to issue a SEND\_DATA verb for the RQSDSS (DDM command) and then a SEND\_DATA verb for the OBJDSS (command data).

**Note:** This can be done with one SEND\_DATA verb instead of two. The SCM then issues a RECEIVE\_AND\_WAIT verb to receive the RPYDSS or OBJDSS from the TCM.

A sample protocol sequence is shown in Figure 3-4 (on page 80). SNA LU 6.2 functions provide an extensive set of functional capabilities that it is impossible to show all the possible sequences. Notes describing key points in the sequence follow the sequence diagram.

The following assumptions have been made for the sample protocol sequence:

- A conversation exists between the SCM and TCM (as described in *APPCMNI* (on page 72) and *SYNCMNI* (on page 931)).
- The response to the RQSDSS is a single RPYDSS.
- No error situations occur.





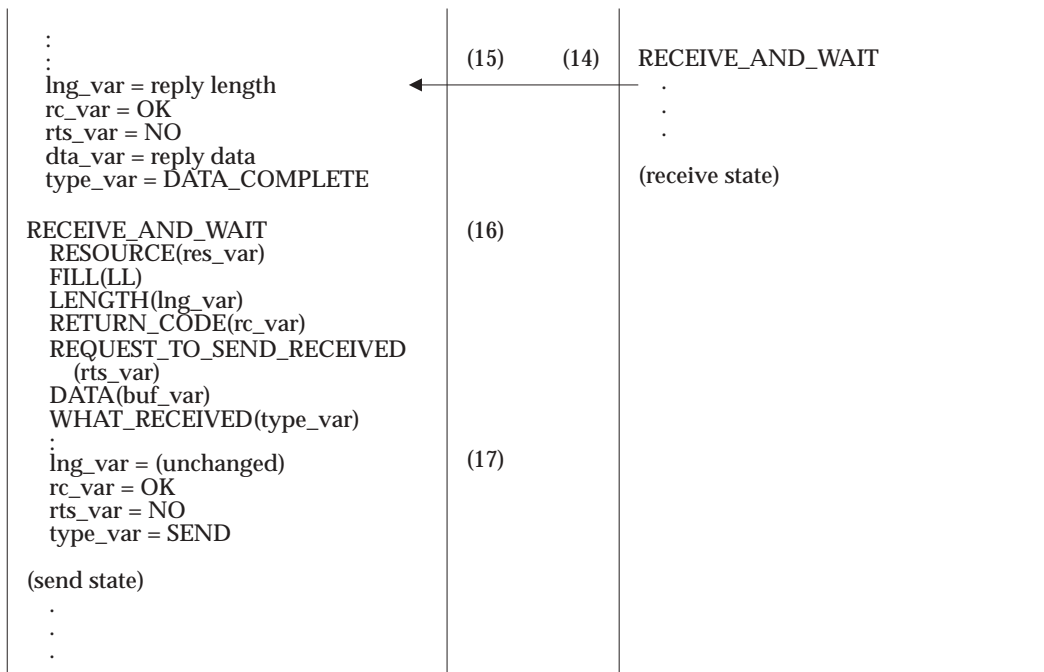


Figure 3-6 Source Sends Command with Data (APPSRCCD) Protocol (Part 3)

**Figure Notes**

1. The TCM issues a RECEIVE\_AND\_WAIT verb to receive the next RQSDSS (DDM command) from the SCM. For this example, a MODREC command is assumed.  
 The FILL parameter indicates that the TCM can receive only one logical record (one DSS structure) from the SNA LU 6.2 communications facility. The FILL parameter could have specified BUFFER instead of LL, but here LL illustrates all receive operations.
2. The SCM issues a SEND\_DATA verb to send an RQSDSS (DDM command).  
 The RESOURCE parameter identifies the conversation resource to be used. This is the same value that was returned to the SCM when the ALLOCATE verb was issued (see APPCMNI (on page 72)). The DATA parameter specifies the location of the data (DDM command) to be sent. The LENGTH parameter gives the total length of the data at the DATA location.
3. The SNA LU 6.2 communications facility accepts the data and queues it for output. Note that the data is not sent to the target system at this time. The SNA LU 6.2 communications facility may delay sending the data until its transmit buffer is full.  
 When the SNA LU 6.2 communications facility has successfully received the data, it sets the RETURN\_CODE parameter to OK; it sets the REQUEST\_TO\_SEND\_RECEIVED parameter to NO, and it returns control to the SCM.
4. The SCM issues a SEND\_DATA verb to send the OBJDSS (command data) that relates to the previous RQSDSS.  
 The DATA parameter specifies the location of the data (OBJDSS) being sent. The LENGTH parameter gives the total length of the data at the DATA location.



5. The SNA LU 6.2 communications facility accepts the data and queues it for output. Note that the data still is not sent to the target system at this time if the SNA LU 6.2 transmit buffer is still not full.
6. The SCM issues a RECEIVE\_AND\_WAIT verb to receive the RPYDSS from the TCM for the last RQSDSS that was sent. The SCM does not receive control back from the SNA LU 6.2 communications facility until the target system responds.

The RESOURCE parameter identifies the conversation resource to be used. The FILL parameter specifies that the SCM is requesting a single logical record. The LENGTH parameter specifies the total length of the buffer space available at the DATA parameter location. Note that the buffer space must be large enough to receive the largest anticipated reply from the TCM.

7. The SCM SNA LU 6.2 facility sends all of the buffered data to the TCM. Once the target SNA LU 6.2 facility receives data, the TCM RECEIVE\_AND\_WAIT verb is completed, and control is return to the TCM.

The contents of the transmission would be:

- Chaining information and a change direction indication
- A logical record containing the RQSDSS (DDM command)
- A logical record containing the OBJDSS (command data)

When the RECEIVE\_AND\_WAIT verb completes its operation, the SNA LU 6.2 communications facility:

- Sets the LENGTH parameter to the length of data placed at the DATA parameter location
- Sets the RETURN\_CODE parameter to OK
- Sets the REQUEST\_TO\_SEND\_RECEIVED parameter to NO
- Sets the WHAT\_RECEIVED parameter to DATA\_COMPLETE

The SCM is now in receive state.

8. After the DDM command is passed to the target agent, the TCM issues another RECEIVE\_AND\_WAIT verb to receive the OBJDSS (command data) associated with the DDM command.

The parameters have the same meaning as for the previous RECEIVE\_AND\_WAIT verb.

9. The target SNA LU 6.2 facility already has the next logical record in its receive buffers and returns it and control to the TCM.

The SNA LU 6.2 facility:

- Sets the LENGTH parameter to the length of the data placed at the DATA parameter location
- Sets the RETURN\_CODE parameter to OK
- Sets the REQUEST\_TO\_SEND\_RECEIVED parameter to NO
- Sets the WHAT\_RECEIVED parameter to DATA\_COMPLETE

10. The TCM processes the OBJDSS, passes control to the command target (on request), and then issues another RECEIVE\_AND\_WAIT verb to check if any more logical records are available to receive.

11. The target SNA LU 6.2 facility does not have any more data in its receive buffers. Therefore, it reports that the TCM has now entered the send state.

The SNA LU 6.2 communications facility does not change the LENGTH parameter; it sets the RETURN\_CODE parameter to OK, sets the REQUEST\_TO\_SEND\_RECEIVED parameter to NO, and sets the WHAT\_RECEIVED parameter to SEND.

12. The TCM issues a SEND\_DATA verb to send the RPYDSS (command reply) to the SCM.
13. The SNA LU 6.2 communications facility accepts the command reply and queues it for output. Nothing is sent at this time.

The SNA LU 6.2 communications facility returns control to the TCM after setting the RETURN\_CODE parameter to OK.

14. The TCM issues a RECEIVE\_AND\_WAIT verb to receive the next RQSDSS (DDM command) from the SCM. The SNA LU 6.2 communications facility sends all of the data in its transmit buffers to the source system.

The contents of the transmission would be:

- Chaining information and a change direction indication
- A logical record containing the RPYDSS (DDM command reply)

The TCM is now in receive state.

15. The receipt of the data causes the RECEIVE\_AND\_WAIT verb to complete its operation at the SCM.

When the RECEIVE\_AND\_WAIT verb completes its operation, the SNA LU 6.2 communications facility:

- Sets the LENGTH parameter to the length of data placed at the DATA parameter location
- Sets the RETURN\_CODE parameter to OK
- Sets the REQUEST\_TO\_SEND\_RECEIVED parameter to NO
- Sets the WHAT\_RECEIVED parameter to DATA\_COMPLETE

The SCM processes the RPYDSS and passes the command reply to the source agent.

16. The SCM issues another RECEIVE\_AND\_WAIT verb to check if any more logical records are available to receive. The parameters are the same as before.
17. The SNA LU 6.2 communications facility does not have any more logical records in its receive buffer and therefore notifies the SCM that it is in send state (the WHAT\_RECEIVED parameter set to SEND).

This completes the sequence. The SCM and TCM are in the same communications states as when the sequence started. This sequence can be repeated many times.

**SEE ALSO**

**Semantic**

*CMNAPPC* (on page 184)

*CMNSYNCPT* (on page 202)

**NAME**

APPSRCCR — LU 6.2 Source Command Returning Data

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

LU 6.2 Source Command Returning Data (APPSRCCR) illustrates the communications sequence occurring when a source communications manager (SCM) sends a single RQSDSS (DDM command) to the target communications manager (TCM) that results in one or more OBJDSSs (reply data string structure objects) returned to the SCM.

This is the normal protocol that is followed when the source agent retrieves records from a remote file or retrieves answer set data from a relational database. The source agent sends a DDM command requesting that the target agent send data to the source agent. The DDM command could be a GETREC, SETxxx, OPNQRY, or CNTQRY command.

A sample protocol sequence is shown in Figure 3-7 (on page 87). SNA LU 6.2 functions provide such an extensive set of functional capabilities that it is impossible to show all the possible sequences. Notes describing key points in the sequence follow the sequence diagram.

The following assumptions have been made for the sample protocol sequence:

- A conversation is successfully established between the SCM and TCM (as described in *APPCMNI* (on page 72) and *SYNCMNI* (on page 931)).
- No error situations occur.



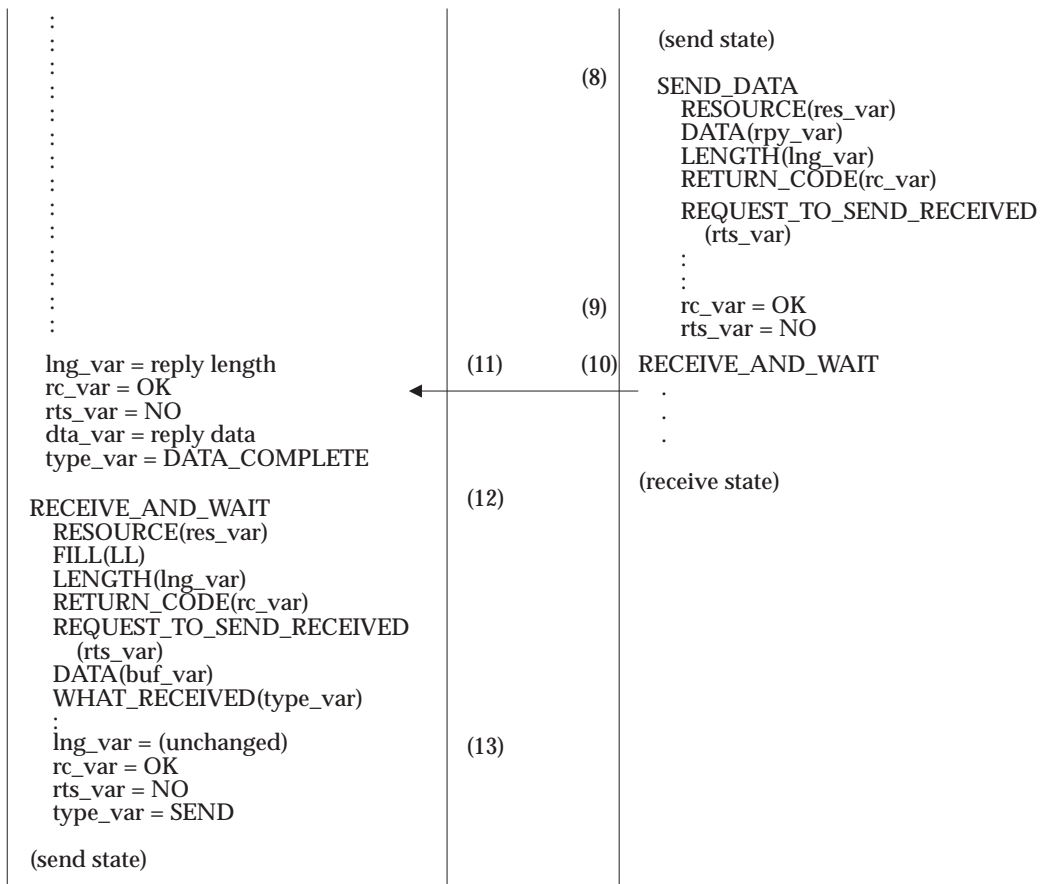


Figure 3-8 Source Sends Command with No Data (APPSRCCR) Protocol (Part 2)

**Figure Notes**

1. The TCM issues a RECEIVE\_AND\_WAIT verb to receive the next RQSDSS (DDM command) from the SCM.

The FILL parameter indicates that the TCM can receive only one logical record (one DDM structure) from the SNA LU 6.2 communications facility. The FILL parameter could have specified BUFFER instead of LL. However, here LL illustrates all receive operations.

2. The SCM issues a SEND\_DATA verb to send an RQSDSS (DDM command).

The RESOURCE parameter identifies the conversation resource to be used. This is the same as the value that was returned to the SCM when the ALLOCATE verb was issued (see APPCMNI (on page 72)). The DATA parameter specifies the location of the data (RQSDSS) to be sent. The LENGTH parameter gives the total length of the data at the DATA location.

3. The SNA LU 6.2 facility accepts the data and queues it for output. Note that the data is not sent to the target system at this time. The SNA LU 6.2 facility might delay sending the data until its transmit buffer is full.

When the SNA LU 6.2 communications facility successfully receives the data, it sets the RETURN\_CODE parameter to OK and the REQUEST\_TO\_SEND\_RECEIVED parameter to NO, and it returns control to the SCM.

4. The SCM issues a RECEIVE\_AND\_WAIT verb to receive the OBJDSS (reply data) from the TCM for the last RQSDSS (DDM command) that was sent. The SCM does not receive control back from the SNA LU 6.2 communications facility until the target server responds.

The RESOURCE parameter identifies the conversation resource to be used. The FILL parameter specifies that the SCM can receive a single logical record. The LENGTH parameter specifies the total length of the buffer space available at the DATA parameter location. Note that the buffer space must be large enough to receive the largest anticipated reply from the TCM.

5. The SCM SNA LU 6.2 communications facility sends all of the buffered data to the target system. Once the target SNA LU 6.2 communications facility receives the data, the TCM RECEIVE\_AND\_WAIT verb completes and returns control to the TCM process.

The contents of the transmission would be:

- Chaining information and a change direction indication
- A logical record containing the RQSDSS (DDM command)

When the RECEIVE\_AND\_WAIT verb completes its operation, the SNA LU 6.2 communications facility:

- Sets the LENGTH parameter to the length of data placed at the DATA parameter location
- Sets the RETURN\_CODE parameter to OK
- Sets the REQUEST\_TO\_SEND\_RECEIVED parameter to NO
- Sets the WHAT\_RECEIVED parameter to DATA\_COMPLETE

The SCM is now in the receive state.

6. After processing the RQSDSS and passing the DDM command to the target agent, the TCM issues another RECEIVE\_AND\_WAIT verb to receive the next RQSDSS.

The parameters have the same meaning as for the previous RECEIVE\_AND\_WAIT verb.

7. The target SNA LU 6.2 communications facility has no more logical records in its receive buffers. Therefore, it reports that the TCM has now entered the send state.

The SNA LU 6.2 communications facility:

- Does not change the LENGTH parameter
- Sets the RETURN\_CODE parameter to OK
- Sets the REQUEST\_TO\_SEND\_RECEIVED parameter to NO
- Sets the WHAT\_RECEIVED parameter to SEND

The TCM is now in the send state.

8. The TCM issues a SEND\_DATA verb to send the OBJDSS (reply data consisting of one or more data records) to the SCM.
9. The target SNA LU 6.2 communications facility accepts the data and queues it for output. Nothing is sent at this time.

The SNA LU 6.2 communications facility returns control to the TCM and sets the RETURN\_CODE to OK and the REQUEST\_TO\_SEND\_RECEIVED to NO.

10. The TCM issues a RECEIVE\_AND\_WAIT verb to receive the next RQSDSS (DDM command) from the SCM. The SNA LU 6.2 communications facility sends all of the data in its transmit buffers to the source system.

The contents of the transmission would be:

- Chaining information and a change direction indication
- A logical record containing the OBJDSS (DDM reply data)

The TCM is now in the receive state.

11. The receipt of the data causes the RECEIVE\_AND\_WAIT verb to complete its operation at the SCM.

When the RECEIVE\_AND\_WAIT verb completes, the SNA LU 6.2 communications facility:

- Sets the LENGTH parameter to the length of the data placed at the DATA parameter location
- Sets the RETURN\_CODE parameter to OK
- Sets the REQUEST\_TO\_SEND\_RECEIVED parameter to NO
- Sets the WHAT\_RECEIVED parameter to DATA\_COMPLETE

The SCM then processes the RPYDSS and passes the reply data (on request) to the source agent.

12. The SCM issues another RECEIVE\_AND\_WAIT verb to check if more logical records are available to receive. The parameters are the same as before.
13. The SNA LU 6.2 communications facility does not have any more logical records in its receive buffer. Therefore, it notifies the SCM that it is in the send state (the WHAT\_RECEIVED parameter set to SEND).

This completes the sequence. The SCM and TCM are in the same communications states as when the sequence started. This sequence can be repeated many times.

#### SEE ALSO

<b>Semantic</b>	<i>APPTRGER</i> (on page 95)
	<i>CMNAPPC</i> (on page 184)
	<i>CMNSYNCPT</i> (on page 202)



**NAME**

APPSRCER — LU 6.2 Source Detected Error

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

LU 6.2 Source Detected Error (APPSRCER) illustrates the communication sequence occurring when a source DDM manager detects an error. The DDM server detects these errors above the SNA LU 6.2 communications facility.

When the source communications manager (SCM) processes the DSSs received from the target communications manager (TCM), the SCM, the source agent, or some other source manager that prevents the DSS contents from being processed might detect an error. The error might be that the TCM sent structures that the SCM did not expect.

A sample protocol sequence is shown in Figure 3-9 (on page 92). SNA LU 6.2 functions provide such an extensive set of functional capabilities that it is impossible to show all the possible sequences. Notes describing key points in the sequence follow the sequence diagram.

The following assumptions have been made for the sample protocol sequence:

- A conversation is successfully established between the SCM and TCM (as described in *APPCMNI* (on page 72) and *SYNCMNI* (on page 931)).

The initial conditions for this sequence are that:

- The SCM has issued a RECEIVE\_AND\_WAIT verb and is waiting to receive either an RPYDSS or OBJDSS from the TCM.
- The TCM has processed the last RQSDSS received from the SCM and issued a SEND\_DATA verb to transmit the RPYDSS to the SCM. However, the TCM has built the RPYDSS structure incorrectly.

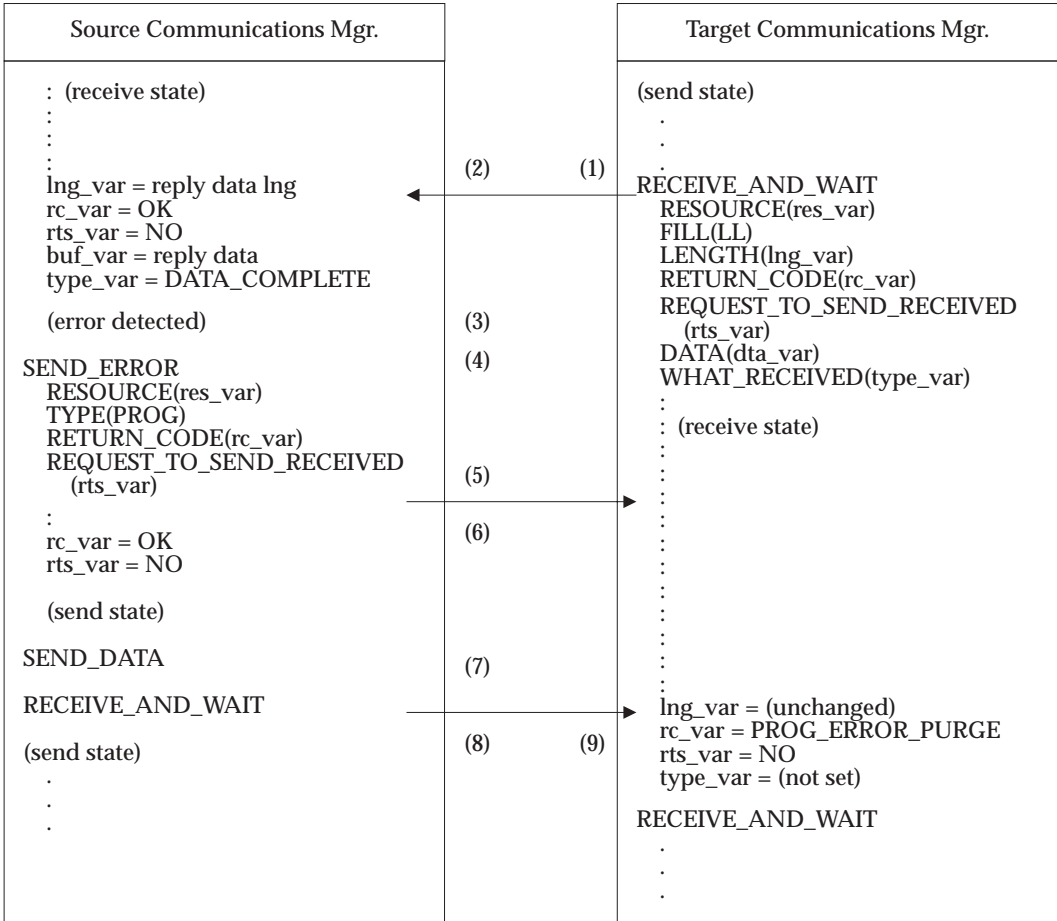


Figure 3-9 Source System Detects Error (APPSRCER) Protocol

Figure Notes

- The TCM issues a RECEIVE\_AND\_WAIT verb to receive the next RQSDSS from the SCM. The TCM does not receive control back from the SNA LU 6.2 communications facility until something has been received.

The RESOURCE parameter identifies the conversation resource being used. The FILL parameter specifies that the TCM wants to receive a single logical record. The FILL parameter could have specified BUFFER instead of LL, but LL is being used for illustrative purposes. The LENGTH parameter specifies the total length of the buffer space available at the DATA parameter location. Note that the buffer space should be large enough to receive the largest anticipated RQSDSS from the SCM.

The contents of the transmission would be:

- Chaining information and a change direction indication

- A logical record containing the incorrect RPYDSS

When the RECEIVE\_AND\_WAIT verb completes its operation, the SNA LU 6.2 communications facility:

- Sets the LENGTH parameter to the length of data placed at the DATA parameter location
- Sets the RETURN\_CODE parameter to OK
- Sets the REQUEST\_TO\_SEND\_RECEIVED parameter to NO
- Sets the WHAT\_RECEIVED parameter to DATA\_COMPLETE

The TCM is now in the receive state.

3. The SCM processes the RPYDSS. However, in doing so the SCM encounters an error that prevents it from successfully processing the RPYDSS.
4. The SCM issues a SEND\_ERROR verb so that the TCM can be notified that the SCM has encountered an error. When the SEND\_ERROR verb completes its operation, the SCM is in the send state.

The RESOURCE parameter identifies the conversation resource to be used. The TYPE parameter specifies that the error is a program error.

5. The source SNA LU 6.2 communications facility sends a negative response to the target system to indicate that the last transmission was in error. The negative response sense code is 0846.

Data still in the source SNA LU 6.2 receive buffers is thrown away, and the SCM is placed in the send state.

The source SNA LU 6.2 communications facility builds an FM header type 7 (FMH7) with sense data 08890000. This is placed in the source SNA LU 6.2 transmit buffers, but it is not sent until later.

6. The SEND\_ERROR verb completes its operation, and control is returned to the SCM. The RETURN\_CODE parameter is set to OK, and the REQUEST\_TO\_SEND\_RECEIVED is set to NO.
7. The SCM's actions after reaching the send state is not architected. Some possibilities are:
  - The SCM can issue a SEND\_DATA to repeat the last RQSDSS.
  - The SCM can attempt some form of backout.
  - The SCM can simply send new RQSDSSs.

The SEND\_DATA verb is shown here without all of its parameters.

**Note:** The SCM is not able to tell the TCM what specific error was detected. The SCM cannot send RPYDSSs to the TCM.

8. When the SCM issues a RECEIVE\_AND\_WAIT verb, the SNA LU 6.2 communications facility transmits everything in its transmit buffers. This includes the FMH7 that was queued from the SEND\_ERROR verb.
9. The TCM RECEIVE\_AND\_WAIT verb completes its operation, and control is returned to the TCM. However, this is not a normal completion.

The RETURN\_CODE is set to PROG\_ERROR\_PURGING to indicate that the SCM detected a program error and purged all data in its receive buffers. The REQUEST\_TO\_SEND\_RECEIVED parameter is set to NO, and the LENGTH parameter is

unchanged.

The TCM is not required to respond to this error indication. The source system is responsible for performing any recovery actions that are necessary. The TCM can log this error or notify someone on the local system, but it is not required.

The TCM process should issue another RECEIVE\_AND\_WAIT verb to receive the next RQSDSS.

**SEE ALSO**

**Semantic**            *CMNAPPC* (on page 184)  
                          *CMNSYNCPT* (on page 202)

**NAME**

APPTRGER — LU 6.2 Target Detected Error

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

LU 6.2 Target Detected Error (APPTRGER) illustrates the communications sequence occurring when a target DDM manager detects an error and reports it to the source communications manager (SCM). The DDM server detects these errors above the SNA LU 6.2 communications facility. This discussion assumes that the detected error results in a reply message with SVRCOD(ERROR) or higher, and if an RQSDSS chain is being processed, error continuation is not being performed.

**Note:** If error continuation is being performed for the detected error, then the communications sequence is similar to that documented in APPSRCCR where the SEND\_DATA verb sends all the RPYDSSs (and OBJDSSs) resulting from the RQSDSS chain (SEND\_ERROR verb is not issued).

When the target communications manager (TCM) processes the data structures received from the SCM, an error might be detected that prevents the TCM or target agent from processing the data structure. The error might be that the SCM sent structures the TCM did not expect.

A sample protocol sequence is shown in Figure 3-10 (on page 96). SNA LU 6.2 functions provide such an extensive set of capabilities that it is impossible to show all the possible sequences. Notes describing key points in the sequence follow the sequence diagram.

The following assumptions has been made for the sample protocol sequence:

- A conversation has been successfully established between the SCM and TCM (as described in *APPCMNI* (on page 72) and *SYNCMNI* (on page 931)).

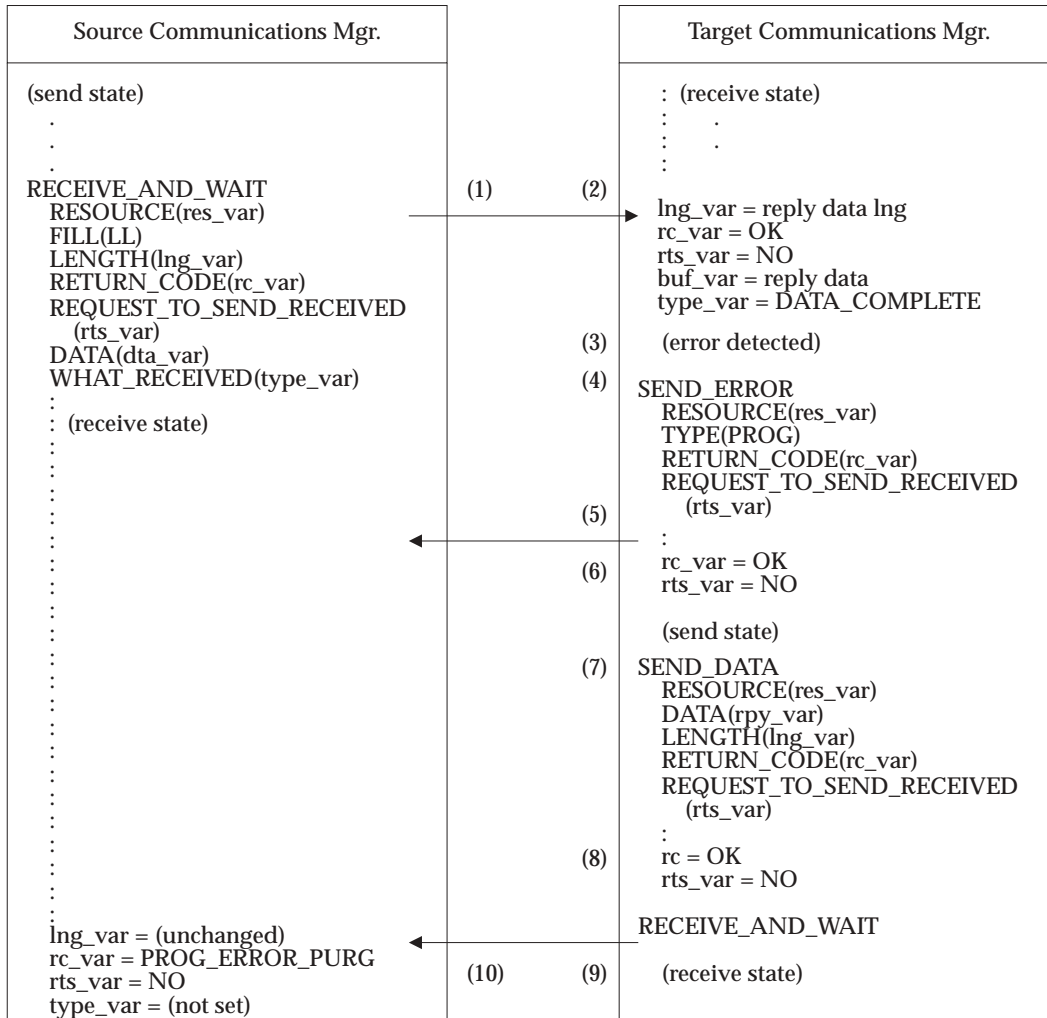
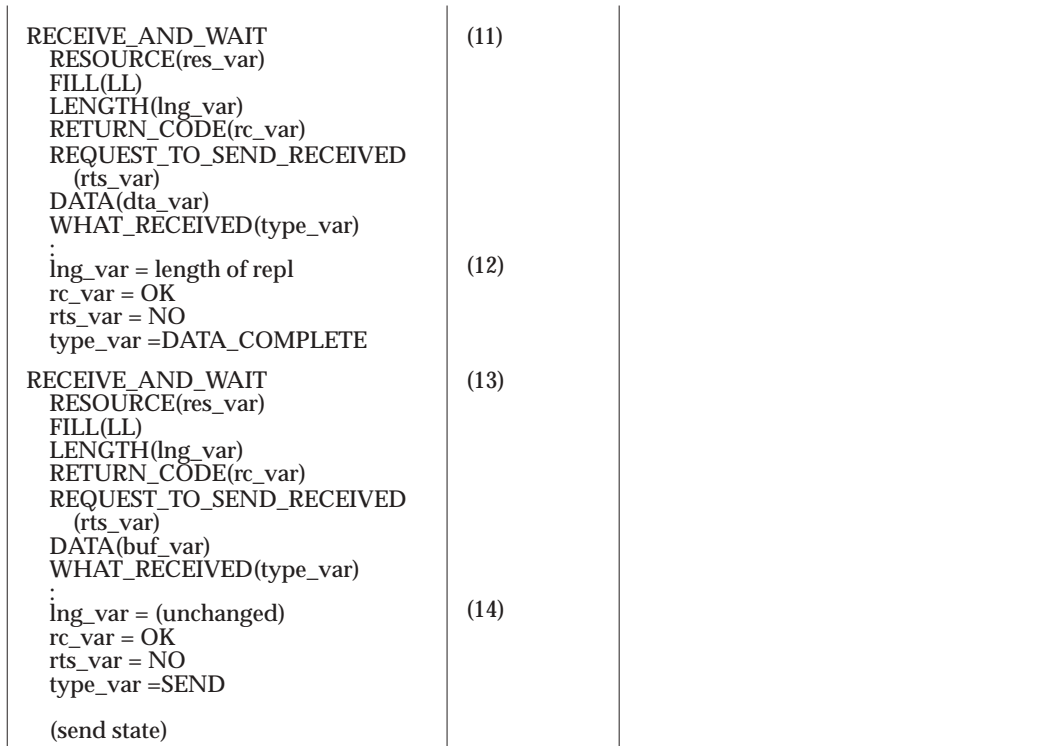


Figure 3-10 Target System Detects Error (APPTRGER) Protocol (Part 1)



**Figure 3-11** Target System Detects Error (APPTRGER) Protocol (Part 2)

**Figure Notes**

1. The SCM issues a RECEIVE\_AND\_WAIT verb and is waiting to receive either an RPYDSS or OBJDSS from the TCM. This causes the SNA LU 6.2 communications facility to transmit all of the data in the SNA LU 6.2 transmit buffers to the target system.

The RESOURCE parameter identifies the conversation resource being used. The FILL parameter specifies that the SCM wants to receive a single logical record. The FILL parameter could have specified FILL(BUFFER) instead of FILL(LL), but here FILL(LL) illustrates all receive operations. The LENGTH parameter specifies the total length of the buffer space available at the DATA parameter location.

2. The SCM SNA LU 6.2 communications facility sends all of the data in its transmit buffers to the TCM. Once the target SNA LU 6.2 facility receives the data, the TCM RECEIVE\_AND\_WAIT verb completes its operation, and control is returned to the TCM.

The content of the transmission would be:

- Chaining information and a change direction indication
- A logical record containing an incorrect RQSDSS

When the RECEIVE\_AND\_WAIT verb completes its operation, the SNA LU 6.2 facility:

- Sets the LENGTH parameter to the length of data placed at the DATA parameter location
- Sets the RETURN\_CODE parameter to OK

- Sets the REQUEST\_TO\_SEND\_RECEIVED parameter to NO
- Sets the WHAT\_RECEIVED parameter to DATA\_COMPLETE

The SCM is now in the receive state.

3. The TCM processes the RQSDSS. In doing so, however, the TCM encounters an error preventing it from successfully processing the RQSDSS.
4. The TCM issues a SEND\_ERROR verb so that the SCM can be notified that the TCM has encountered an error. When the SEND\_ERROR verb completes its operation, the TCM is in the send state.

The RESOURCE parameter identifies the conversation resource being used. The TYPE parameter specifies that the error is a program error.

5. The target SNA LU 6.2 communications facility sends a negative response to the source system to indicate that the last transmission was in error. The negative response sense code is 0846.

Data still in the target SNA LU 6.2 communications facility receive buffers is thrown away and the TCM is placed in the send state.

The target SNA LU 6.2 communications facility builds an FM header type 7 (FMH7) with sense data 08890000. This is placed in the target SNA LU 6.2 communications facility transmit buffers, but it is not sent until later.

6. The SEND\_ERROR verb completes its operation, and control is returned to the TCM. The RETURN\_CODE parameter is set to OK, and the REQUEST\_TO\_SEND\_RECEIVED parameter is set to NO.
7. The TCM issues a SEND\_DATA verb to send the RPYDSS (command reply) that indicates the specific error condition encountered.

The RESOURCE parameter identifies the conversation resource to be used. The DATA parameter specifies the location of the data (RPYDSS) to be sent. The LENGTH parameter gives the total length of the data at the DATA location.

8. The SNA LU 6.2 communications facility accepts the data and queues it for output. The data is not sent to the source system at this time. The SNA LU 6.2 communications facility may delay sending the data until its transmit buffer is full.

When the SNA LU 6.2 communications facility successfully receives the data, the RETURN\_CODE parameter is set to OK, the REQUEST\_TO\_SEND\_RECEIVED parameter is set to NO, and control is returned to the TCM.

9. The TCM issues a RECEIVE\_AND\_WAIT verb to receive the next RQSDSS from the SCM. The SNA LU 6.2 communications facility sends all of the data in its transmit buffers to the source system.

The contents of the transmission would be:

- Chaining information and a change direction indication
- FMH7(08890000)
- A logical record containing the RPYDSS

The TCM is now in the receive state.

10. Once the SNA LU 6.2 communications facility receives the data, the RECEIVE\_AND\_WAIT verb completes at the SCM. However, the RETURN\_CODE parameter indicates that the



RECEIVE\_AND\_WAIT verb did not complete its operation normally. This tells the SCM that the TCM encountered an error.

When the RECEIVE\_AND\_WAIT verb completes its operation, the SNA LU 6.2 communications facility does not change the LENGTH parameter; it sets the RETURN\_CODE parameter to PROG\_ERROR\_PURGE, and sets the REQUEST\_TO\_SEND\_RECEIVED parameter to NO.

11. The RETURN\_CODE of PROG\_ERROR\_PURGE tells the SCM that the TCM encountered an error and threw away data that was in its SNA LU 6.2 communications facility receive buffers. This return code does not tell the SCM or source agent what specific error the TCM encountered. Therefore, the SCM issues another RECEIVE\_AND\_WAIT verb to receive the RPYDSS (command reply). The parameters are the same as before.
12. The SNA LU 6.2 communications facility already has the next logical record in its receive buffers, so the RECEIVE\_AND\_WAIT verb is completed and control is returned to the SCM.

When the RECEIVE\_AND\_WAIT verb completes its operation, the SNA LU 6.2 communications facility:

- Sets the LENGTH parameter to the length of data placed at the DATA parameter location
- Sets the RETURN\_CODE parameter to OK
- Sets the REQUEST\_TO\_SEND\_RECEIVED parameter to NO
- Sets the WHAT\_RECEIVED parameter to DATA\_COMPLETE

The SCM then processes the RPYDSS and passes it to the source agent.

13. The SCM issues another RECEIVE\_AND\_WAIT verb to determine if there are more logical records to receive. The parameters are the same as before.
14. The SNA LU 6.2 communications facility has no more logical records in its receive buffer. Therefore, it notifies the SCM process that it is in the send state (the WHAT\_RECEIVED parameter set to SEND).

The source agent is responsible for performing any recovery actions that are necessary. The TCM might want to log the error or notify someone on the local system, but it is not required.

#### SEE ALSO

<b>Semantic</b>	<i>CMNAPPC</i> (on page 184)
	<i>CMNSYNCPT</i> (on page 202)

**NAME**

ARMCORR — ARM Correlator

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2161'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

The ARM Correlator (ARMCORR) string specifies a token that is conveyed between source and target servers for associating contextual information with all SQL requests for a connection in an environment with ARM application instrumentation. For more information, see ARM 4.0 and the DRDA Reference.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'2161'	
value	INSTANCE_OF	BYTSTRDR - Byte String
	MINLEN	1
	MAXLEN	255
	MINLVL	7
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** ACCRDB (on page 42)

**NAME**

ARRAY — Object Array

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'004B'

**Length** \*

**Class** CLASS

**Sprcls** COLLECTION - Collection Object

Object Array (ARRAY) Collection Object is a collection of objects in which each element is associated with an ordinal number that specifies its position in the array.

An array can be constrained to contain:

- Only objects of a specified class
- Only objects of a specified class and length
- Only objects of a specified set of classes

Constraints on arrays are specified:

- By defining a subclass of ARRAY and specifying the class or length as an attribute of its value variable
- By defining a variable to be of class ARRAY and specifying the ELMCLS (element of class) attribute as an additional attribute of the variable

**Literal Form**

The literal form of an ARRAY is:

#(element...)

where element is a literal number, string, array, or other literal. Embedded arrays are not preceded by #.

An example of an unconstrained array is:

#1 'A' (1 2 3) 'food')

An example of a class constrained array is (all elements are class number):

#(1 2 3 4 5)

An example of a class and length constrained array is (all elements are class STRING and LENGTH=6):

#('apple ' 'orange' 'plum ')

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
<b>length</b>	*

---

<b>class</b>	<b>X'004B'</b>	
<b>value</b>	<b>REPEATABLE</b>	
	<b>SPRCLS</b>	<b>OBJECT - Self-identifying Data</b>
	<b>NOTE</b>	<b>An array can have zero or more elements.</b>
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

---

**SEE ALSO**

None.

**NAME**

ASSOCIATION — Name with Value Association

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'0001'

**Length** \*

**Class** CLASS

**Sprcls** ORDCOL - Ordered Collection

Name with Value Association (ASSOCIATION) Collection Object specifies the association of a name with an object that is treated as the value of the name.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'0001'	
key	SPRCLS	OBJECT - Self-identifying Data
value	SPRCLS	OBJECT - Self-identifying Data
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**Semantic** *DCTINDEN* (on page 259)  
*DEFINITION* (on page 267)  
*INHERITANCE* (on page 437)  
*QLFATT* (on page 664)

**NAME**

ATMIND — Atomicity Indicator

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2159'**Length** \***Class** CLASS**Sprcls** SCALAR - Scalar Object

Atomicity Indicator (ATMIND) specifies the atomic property of an EXCSQLSTT command. It is used to indicate whether the EXCSQLSTT command is any of the following:

- An atomic operation
- A non-atomic multi-row input operation

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	5	
<b>class</b>	X'2159'	
<b>value</b>	INSTANCE_OF	BYTDR - An 8-bit Data Representation Value
	ENUVAL	X'00' - EXCSQLSTT command s an atomic operation.
	ENUVAL	X'01' - EXCSQLSTT command is a non-atomic multi-row input operation.
	OPTIONAL	
	DFTVAL	X'00' - EXCSQLSTT command is an atomic operation.
	MINLVL	7
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *BYTDR* (on page 145)  
*DFTVAL* (on page 276)  
*EXCSQLSTT* (on page 381)  
*MINLVL* (on page 518)

**Semantic** *CLASS* (on page 158)  
*EXCSQLSTT* (on page 381)  
*QDDRDBD* (on page 657)  
*SCALAR* (on page 796)

**NAME**

ATTLLST — Attribute List

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'0046'

**Length** \*

**Class** CLASS

**Sprcls** COLLECTION - Collection Object

Attribute List (ATTLLST) Collection Object specifies a list of attributes.

Attribute lists specify the attributes of a single variable, value, or another aspect of an object.

An attribute is an object whose codepoint (or term name) provides a name for a characteristic and whose value qualifies the characteristic. For example, LENGTH 32.

This is an attribute with name LENGTH and value 32. This attribute asserts that the length of the term is 32. Nothing is specified regarding the units of length of the term, only that there are (or must be) 32 units. Other attributes must be specified to determine whether the units are bits, bytes, or some other unit.

All attribute names and values are either dictionary terms or literals. For example, INSTANCE\_OF BINDR.

This is an attribute that specifies that the term described must be an instance of the class BINDR. The dictionary term INSTANCE\_OF describes the concept of instances of classes. The dictionary term BINDR describes BINDR data.

A single attribute specification is generally not sufficient to describe a term. In the above two examples, LENGTH 32 and INSTANCE\_OF BINDR, these specify two attributes, but only when they are used together is it known that an instance of the class BINDR that is 32 bits long is being described. The unit of length is inferred to be bits because that is how BINDR values are represented, as the term BINDR describes.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'0046'	
attribute	SPRCLS REPEATABLE NOTE	OBJECT - Self-identifying Data
	NOTE	Each attribute is a separate object in the attribute list.
	NOTE	Each field attribute can be specified only once in a declaration of a field.
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

<b>insvar</b>	<i>DEFINITION</i> (on page 267) <i>QLFATT</i> (on page 664)
<b>Semantic</b>	<i>DEFLST</i> (on page 268) <i>QLFATT</i> (on page 664)



**NAME**

BGNATMCHN — Begin Atomic Chain

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD  
**Codepoint** X'1803'  
**Length** \*  
**Class** CODPNT  
**Sprcls** COMMAND - Command

The Begin Atomic Chain (BGNATMCHN) command is used to initiate an atomic chain of Execute SQL Statement (EXCSQLSTT) commands. The chain is atomic because in the event that any one of the EXCSQLSTT commands fails, then all other changes made to the database within this chain will be undone.

The chain is terminated by the End Atomic Chain (ENDATMCHN) command.

An empty chain—that is, one without any EXCSQLSTT commands between the BGNATMCHN and ENDATMCHN commands—is allowed.

**Source System Processing**

The BGNATMCHN command is sent using a Data Stream Structure Format (DSSFMT) with a *dstyp* value of 5 to indicate that no reply is expected if the command is processed successfully by the target system.

**DSS Carrier: RQSDSS**

**Target System Processing**

The optional *rtncstt* parameter specifies whether upon successful processing of each EXCSQLSTT command within the atomic chain the target server must return one or more SQLSTT reply data objects, each containing an SQL SET statement for a special register whose setting has been modified on the current connection, if any special register has had its setting modified during execution of the EXCSQLSTT command.

Normal completion results in no reply being sent back for this command.

**Exceptions**

Exception conditions the RDB detects are reported in an SQLCARD object. Other exception conditions are indicated by an error reply message.

**Source System Reply Processing**

Return any error reply messages to the requester.

<b>clsvar</b>	NIL
<b>insvar</b>	CLASS INSTANCE VARIABLES

length	*	
class	X'1803'	
rttnsetstt	INSTANCE_OF OPTIONAL MINLVL	RTNSETSTT - Return SET Statement  7
clscmd	NIL	
inscmd	NIL	
cmddta	NIL	
<b>rpydta</b>		<b>REPLY OBJECTS</b>
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL ENUVAL DFTVAL NOTE	TYPDEFNAM - Data Type Definition Name 'QTDSQL370' 'QTDSQL400' 'QTDSQLX86' 'QTDSQLASC' 'QTDSQLVAX' " The default means the value received on ACCRDBRM is used.
	OPTIONAL	
X'0035'	INSTANCE_OF DFTVAL NOTE	TYPDEFOVR - TYPDEF Overrides " The default means the value received on ACCRDBRM is used.
	OPTIONAL	
X'002F'	INSTANCE_OF OPTIONAL	SQLCARD - SQL Communications Area Reply Data
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported
X'1245'	INSTANCE_OF	PRCCNVRM - Protocol Conversation Error
X'1233'	INSTACNE_OF	RSCLMTRM - Resource Limit Reached
X'2213'	INSTANCE_OF	SQLERRRM - SQL Error Condition
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error

**SEE ALSO**

<b>insvar</b>	<i>EXCSQLSTT</i> (on page 381) <i>PRCCNVCD</i> (on page 621)
<b>rpydta</b>	<i>ACCRDBRM</i> (on page 48) <i>TYPDEFNAM</i> (on page 1027) <i>TYPDEFOVR</i> (on page 1030)
<b>cmdrpy</b>	<i>AGNPRMRM</i> (on page 65) <i>CMDCHKRM</i> (on page 173) <i>CMDNSPRM</i> (on page 176) <i>PRCCNVRM</i> (on page 625) <i>RSCLMTRM</i> (on page 778) <i>SQLERRRM</i> (on page 864) <i>SYNTAXRM</i> (on page 989)

**Semantic**      *CODPNT* (on page 233)  
*COMMAND* (on page 240)  
*DSSFMT* (on page 318)  
*ENDATMCHN* (on page 333)  
*EXCSQLSTT* (on page 381)  
*QDDBASD* (on page 651)  
*RQSDSS* (on page 774)  
*SQLCARD* (on page 855)

## NAME

BGNBND — Begin Binding a Package to an RDB

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'2002'  
**Length** \*  
**Class** CLASS  
**Sprcls** COMMAND - Command

Begin Binding a Package to an RDB (BGNBND) command begins the process of binding a package to a relational database (RDB). The BNDSQLSTT command binds SQL statements and referenced application variable definitions to the package after the BGNBND command has been executed. The process of package binding must be terminated before a subsequent BGNBND, CLSQRY, CNTQRY, DRPPKG, DSCRDBTBL, DSCSQLSTT, EXCSQLIMM, EXCSQLSTT, OPNQRY, PRPSQLSTT, or REBIND command can be issued.

The ENDBND command explicitly terminates package binding while the SQLAM implicitly terminates package binding by commit or rollback processing. Concurrent package binding can occur across multiple RDBs. Each RDB controls its own bind process, beginning with a BGNBND command and ending with either an ENDBND command, or commit or rollback processing.

**Source System Processing**

The source system determines the location of the RDB:

**Local** Call the local RDB server.  
**Remote** Send the BGNBND command to the remote RDB server.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergo conversion between the code pages.

**DSS Carrier: RQSDSS****Target System Processing**

The *bndchkexs* parameter controls whether to treat the absence of a named RDB object or the lack of authorization to a named RDB object as an error during the bind process.

The *bndcrtctl* parameter specifies under what conditions to create a package as part of the bind process.

The *bndexopt* parameter controls whether the target SQLAM causes the target RDB to explain the explainable SQL statements that are bound into the package.

The *decprc* parameter specifies the decimal precision that the target RDB uses during decimal arithmetic processing.

The *dftrdbcol* parameter specifies the default RDB collection identifier that the target RDB uses to perform RDB object name completion functions, if necessary, on SQL statements bound into the

package.

The *dgriopr1* parameter specifies the degree of I/O parallel processing static SQL statements bound to the package use, if the RDB supports I/O parallel processing.

The *pkgathopt* parameter controls whether the existing package authorizations are kept or revoked when an existing package is replaced.

The *pkgathrul* parameter specifies which authorization identifier to use when dynamic SQL in a package is executed. The possible alternatives are either the requester (the person executing the package) or the owner (the person who owns the package).

The *pkgdfcc* parameter specifies the default CCSIDs for any character or graphic column that an SQL CREATE or SQL ALTER table statement defines that does not have an explicit CCSID specified.

The *pkgdfcst* parameter specifies the default SQL character subtype for any character columns that an SQL CREATE or SQL ALTER table statement in which an explicit subtype has not been specified defines.

The *pkgisolvl* parameter specifies the isolation level that the RDB uses when SQL statements in this package are executed unless a target RDB runtime mechanism overrides it.

The *pkgnamct* parameter specifies the fully qualified name and the consistency token being assigned to the new package being bound. No two packages in the same RDB can have the same fully qualified package names and consistency tokens.

The *pkgownid* parameter specifies the end-user name (identifier) of the package's owner.

The *pkgrplopt* parameter specifies what action to take if a package already exists that has the package and version name specified by the *pkgnamct* and *vrsnam* parameters.

The *pkgrplvrs* parameter specifies the version name of the package replaced with the package being bound if the package is bound successfully. This parameter is ignored when PKGRPLOPT(PKGRPLNA) is specified.

The *prpsttkp* parameter specifies whether an application needs to prepare dynamic SQL statements again after a transaction is committed.

The *qryblkctl* parameter specifies the query blocking protocol used by all of the queries in the package.

The *rdbnam* parameter specifies the name of the RDB that the ACCRDB command accesses. This is the RDB to which the package is being bound. If the *rdbnam* parameter is specified, the parameter value must be the same value specified for the *rdbnam* parameter on the ACCRDB command.

The *rdbrlsopt* parameter specifies when the RDB should release the package execution resources for this package.

The *sttdatfmt* parameter specifies the date format used in the SQL statements.

The *sttdecdel* parameter specifies the character used as the decimal delimiter in static SQL statements.

The *sttstrdel* parameter specifies which characters delimit SQL strings and identifiers in static SQL statements.

The *stttimfmt* parameter specifies the time format used in the SQL statements.

The *title* parameter specifies descriptive text associated with the package. The target RDB can ignore this parameter. This means that the target RDB does not have to store or retain the value

of this parameter. The target RDB can also truncate the value of the *title* parameter to conform to any size restrictions that the target RDB might have on the amount of descriptive text that can be associated with a package.

The *vrnam* parameter specifies a version name associated with the package name. No two packages can have the same fully qualified package names and version names in the same RDB. If the version name is not specified, it defaults to null.

Additional bind options may be sent in occurrences of BNDOPT cmd data objects. If the target server does not recognize or cannot process any of the bind options then it responds with a non-zero SQLSTATE in SQLCARD. The SQLERRMC error field in SQLCARD indicates which bind option was rejected. If multiple bind options are rejected, only the first one is reported. The target server has the option of terminating the bind process with an error SQLSTATE or continuing by sending a warning SQLSTATE.

The TYPDEFNAM reply data object specifies the name of the data type to data representation mapping definitions that the target SQLAM uses when sending the SQLCARD object for this command.

The TYPDEFOVR reply data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type to data representation mapping definitions for the SQLCARD object for this command.

The BGNBND command reserves the package and version name, not including the consistency token, so that no concurrent requester can specify the same package and version name on a BGNBND command.

The BGNBND command is usually followed by one or more BNDSQLSTT commands. However, it is valid for an ENDBND command to follow the BGNBND command. In this case, the target RDB must reserve the number of sections that the *maxsctnbr* parameter of the ENDBND command specifies for dynamic SQL statements to use.

If commit processing terminates the bind process, then the maximum section number for the package is assumed to be one of the following:

- The highest section number a BNDSQLSTT command references
- The number one if no BNDSQLSTT commands were sent

If a BNDSQLSTT command specifies a section number that is at least one number greater than the highest section number the previous BNDSQLSTT command specifies, then the target SQLAM must cause sections to be generated for dynamic SQL statements for the unreferenced sections. Or, if the *maxsctnbr* parameter of an ENDBND command specifies a section number that is greater than the highest section number a BNDSQLSTT command references for this bind process, then the target SQLAM must cause sections to be generated (reserved) for dynamic SQL statements for the intervening (unreferenced) sections.

For example, if section 5 was the highest section number a BNDSQLSTT command referenced, and the ENDBND *maxsctnbr* parameter specified a value of ten, then the target SQLAM must cause sections 6, 7, 8, 9, and 10 to be generated (reserved) in the package.

Normal completion of the BGNBND command returns an SQLCARD object. An RDBUPDRM must be returned if the command makes recoverable updates in the RDB, and if the server in the current unit of work (UOW) does not return a prior RDBUPDRM. A recoverable update is an RDB update that writes information to the recovery log of the RDB.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergo conversion between the code pages.

**Exceptions**

Exception conditions that the RDB detects are reported in an SQLCARD object following a BGNBNDRM.

If a BGNBND command is issued before a previous bind process (BGNBND or REBIND command) has been terminated, then the command is rejected with the PKGBPARM.

If the specified RDB is not currently accessed, then the command is rejected with the RDBNACRM.

If the *pkgownid* parameter is specified and the authentication or verification of the end-user name (identifier) fails, then the command is rejected with the BGNBNDRM followed by an SQLCARD object.

**Source System Reply Processing**

Return any reply messages and data objects to the requester.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'2002'	
bndchkexs	INSTANCE_OF OPTIONAL	BNDCHKEXS - Bind Existence Checking
bndcrtctl	INSTANCE_OF OPTIONAL	BNDCRTCTL - Bind Package Creation Control
bndexpopt	INSTANCE_OF OPTIONAL	BNDEXPOPT - Bind Explain Option
decprc	INSTANCE_OF OPTIONAL	DECPRC - Decimal Precision
dftrdbcol	INSTANCE_OF OPTIONAL	DFTRDBCOL - Default RDB Collection Identifier
dgriopr1	INSTANCE_OF OPTIONAL IGNORABLE MINLVL	DGRIOPRL - Degree of I/O Parallelism 4
pkgathopt	INSTANCE_OF	PKGATHOPT - Package Authorization Option

	OPTIONAL	
pkgathrul	INSTANCE_OF OPTIONAL IGNORABLE MINLVL	PKGATHRUL - Package Authorization Rules  5
pkgdftcc	INSTANCE_OF OPTIONAL	PKGDFTCC - Package Default CCSIDs for a Column
pkgdftcst	INSTANCE_OF OPTIONAL	PKGDFTCST - Package Default Character Subtype
pkgisolvl	INSTANCE_OF REQUIRED	PKGISOLVL - Package Isolation Level
pkgnamct	INSTANCE_OF REQUIRED	PKGAMCT - RDB Package Name and Consistency Token
pkgownid	INSTANCE_OF OPTIONAL	PKGOWNID - Package Owner Identifier
pkgrplopt	INSTANCE_OF OPTIONAL	PKGRPLOPT - Package Replacement Option
pkgrplvrs	INSTANCE_OF OPTIONAL DFTVAL NOTE	PKGRPLVRS - Replaced Package Version Name  " If this parameter is not specified, then its default value is the value that the <i>vrsnam</i> parameter on this command assumes. This parameter is ignored when PKGRPLOPT(PKGRPLNA) is specified.
prpsttkp	INSTANCE_OF OPTIONAL MINLVL	PRPSTTKP - Package Owner Identifier  7
qryblkctl	INSTANCE_OF OPTIONAL	QRYBLKCTL - Query Block Protocol Control
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
rdbrlsopt	INSTANCE_OF OPTIONAL	RDBRLSOPT - RDB Release Option
sttdatfmt	INSTANCE_OF OPTIONAL	STTDATFMT - Statement Date Format
sttdecdel	INSTANCE_OF OPTIONAL	STTDECDEL - Statement Decimal Delimiter
sttstrdel	INSTANCE_OF OPTIONAL	STTSTRDEL - Statement String Delimiter
stttimfmt	INSTANCE_OF OPTIONAL	STTTIMFMT - Statement Time Format
title	INSTANCE_OF OPTIONAL IGNORABLE	TITLE - Title



vrsnam	INSTANCE_OF OPTIONAL	VRSNAM - Version Name
clscmd	NIL	
inscmd	NIL	
<b>cmddta</b>		<b>COMMAND OBJECTS</b>
X'2405'	INSTANCE_OF OPTIONAL REPEATABLE MINLVL	BNDOPT - Bind Option  5
<b>rpydta</b>		<b>REPLY OBJECTS</b>
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL MINLVL ENUVAL MINLVL OPTIONAL DFTVAL NOTE	TYPDEFNAM - Data Type Definition Name 'QTDSQL370' 'QTDSQL400' 'QTDSQLX86' 'QTDSQLASC' 4 'QTDSQLVAX' 4 " The default means the value received on ACCRDBRM is used.
X'0035'	INSTANCE_OF OPTIONAL DFTVAL NOTE	TYPDEFOVR - TYPDEF Overrides " The default means the value received on ACCRDBRM is used.
X'1C03'	INSTANCE_OF OPTIONAL MINLVL	MGRLVLOVR - Manager Level Overrides  7
X'2408'	INSTANCE_OF REQUIRED NOTE	SQLCARD - SQL Communications Area Reply Data If the SQLCARD indicates an error condition, it must be preceded by a BGNBNDRM.
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'220D'	INSTANCE_OF	ABNUOWRM - Abnormal End Unit of Work Condition
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'2208'	INSTANCE_OF	BGNBNDRM - Begin Bind Error
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported
X'1253'	INSTANCE_OF	OBJNSPRM - Object Not Supported
X'2209'	INSTANCE_OF	PKGBPARM - RDB Package Binding Process Active
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error

X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'2204'	INSTANCE_OF	RDBNACRM - RDB Not Accessed
X'2218'	INSTANCE_OF MINLVL	RDBUPDRM - RDB Update Reply Message 4
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limits Reached
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'125F'	INSTANCE_OF	TRGNSPRM - Target Not Supported
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

**SEE ALSO**

<b>insvar</b>	<i>STTDECDEL</i> (on page 897) <i>STTSTRDEL</i> (on page 899)
<b>inscmd</b>	<i>SQLAM</i> (on page 847)
<b>Semantic</b>	<i>BNDCHKEXS</i> (on page 125) <i>BNDCRTCTL</i> (on page 127) <i>BNDEXSRQR</i> (on page 131) <i>BNDSQLSTT</i> (on page 136) <i>DFTPKG</i> (on page 273) <i>ENDBND</i> (on page 336) <i>FRCFIXROW</i> (on page 424) <i>OPNQRY</i> (on page 555) <i>PKGATHKP</i> (on page 579) <i>PKGATHOPT</i> (on page 580) <i>PKGATHRVK</i> (on page 584) <i>PKGBNARM</i> (on page 585) <i>PKGBPARAM</i> (on page 586) <i>PKGOWNID</i> (on page 600) <i>PKGRPLALW</i> (on page 602) <i>PKGRPLNA</i> (on page 603) <i>PKGRPLVRS</i> (on page 605) <i>PRPSTTKP</i> (on page 642) <i>QRYBLKCTL</i> (on page 672) <i>RDBCMM</i> (on page 728) <i>RDBRLLBCK</i> (on page 745) <i>REBIND</i> (on page 753) <i>SQL</i> (on page 835) <i>SQLAM</i> (on page 847)

**NAME**

BGNBNDRM — Begin Bind Error

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2208'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

Begin Bind Error (BGNBNDRM) Reply Message indicates that the package binding process could not be initiated because an error condition exists. This reply message is always followed by an SQLCARD object that indicates why the package binding process could not be initiated.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
<b>length</b>	*	
<b>class</b>	X'2208'	
<b>pkgnamct</b>	INSTANCE_OF REQUIRED	PKGNAMECT - RDB Package Name and Consistency Token
<b>rdbnam</b>	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
<b>vrsnam</b>	INSTANCE_OF REQUIRED	VRSNAM - Version Name
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**cmdrpy** *BGNBND* (on page 110)

**rpydta** *BGNBND* (on page 110)

**Semantic** *PKGRPLNA* (on page 603)

**NAME**

BIN — Binary Integer Number

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD  
**Codepoint** X'0003'  
**Length** \*  
**Class** CLASS  
**Sprcls** NUMBER - Number

Binary Integer Number (BIN) is expressed by using an instance of BINDR.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'0003'	
value	INSTANCE_OF NOTE	BINDR - Binary Number Field The LENGTH attribute must be specified for each instance of BIN.
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *DCTINDEN* (on page 259)  
**mgrlvl** *MANAGER* (on page 491)  
**Semantic** *DECPRC* (on page 266)  
*ENULEN* (on page 346)  
*FDODSCOFF* (on page 407)  
*FDODTAOFF* (on page 409)  
*FDOPRMOFF* (on page 413)  
*FDOTRPOFF* (on page 414)  
*INHERITANCE* (on page 437)  
*LENGTH* (on page 472)  
*MAXBLKEXT* (on page 494)  
*MAXLEN* (on page 495)  
*MAXRSLCNT* (on page 497)  
*MAXSCTNBR* (on page 498)  
*MGRVLN* (on page 509)  
*MINLEN* (on page 517)  
*MINLVL* (on page 518)  
*NBRROW* (on page 530)  
*OBJOVR* (on page 544)  
*OOPOVR* (on page 547)  
*PKGATHRUL* (on page 581)  
*QRYBLKSZ* (on page 678)  
*QRYBLKTYP* (on page 679)  
*QRYROWNBR* (on page 694)  
*QRYROWSET* (on page 697)

*RESPKTSZ* (on page 763)  
*SQLSTTNBR* (on page 870)  
*SVRCOD* (on page 911)

**NAME**

BINDR — Binary Number Field  
 Name of term prior to Level 2: BINSTR

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD  
**Codepoint** X'0042'  
**Length** \*  
**Class** CLASS  
**Sprcls** FIELD - A Discrete Unit of Data

Binary Number Field (BINDR) specifies a string of bits that represents a signed binary number.

The high-order bit is the sign of the number:

bit 0-0 = positive sign  
 1 = negative sign

bits 1-n = magnitude of the number, negative magnitudes are stored in "two's complement" form.

B'0000 0000 0000 0001' is a plus 1  
 Bit -> 0000 0000 0011 1111  
 Number 0123 4567 8901 2345

B'1111 1111 1111 1111' is a minus 1  
 Bit -> 0000 0000 0011 1111  
 Number 0123 4567 8901 2345

**Length Specification**

The length that the LENGTH attribute specifies must be the total number of bits used to represent a signed binary number, including the sign bit.

**Literal Form**

The literal form of a binary string is an optionally signed decimal number; for instance, 57 or -15.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
value	INSTANCE_OF	BITSTRDR - Bit String Field
	ENULEN	8
	ENULEN	16
	ENULEN	24
	ENULEN	32
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>vldattls</b>	<b>NIL</b>	

## SEE ALSO

<b>clsvar</b>	<i>CLASS</i> (on page 158)
<b>insvar</b>	<i>BIN</i> (on page 118) <i>CMDSRCID</i> (on page 178) <i>DECPRC</i> (on page 266) <i>DGRIOPRL</i> (on page 279) <i>ENCALG</i> (on page 331) <i>ENCKEYLEN</i> (on page 332) <i>ENULEN</i> (on page 346) <i>FDODSCOFF</i> (on page 407) <i>FDODTAOFF</i> (on page 409) <i>FDOPRMOFF</i> (on page 413) <i>FDOTRPOFF</i> (on page 414) <i>IPADDR</i> (on page 450) <i>LENGTH</i> (on page 472) <i>MAXBLKEXT</i> (on page 494) <i>MAXLEN</i> (on page 495) <i>MAXRSLCNT</i> (on page 497) <i>MAXSCTNBR</i> (on page 498) <i>MGRVLN</i> (on page 509) <i>MINLEN</i> (on page 517) <i>MINLVL</i> (on page 518) <i>NBRROW</i> (on page 530) <i>OBJDSS</i> (on page 536) <i>OBJECT</i> (on page 540) <i>OUTOVROPT</i> (on page 576) <i>PKGATHRUL</i> (on page 581) <i>PKGSN</i> (on page 606) <i>PLGINCNT</i> (on page 610) <i>QRYBLKFACT</i> (on page 675) <i>QRYBLKTYP</i> (on page 679) <i>QRYROWNBR</i> (on page 694) <i>QRYROWSET</i> (on page 697) <i>RPYDSS</i> (on page 766) <i>RQSCRR</i> (on page 772) <i>RQSDSS</i> (on page 774) <i>RESPKTSZ</i> (on page 763) <i>RTNEXTDTA</i> (on page 792) <i>SECMEC</i> (on page 811) <i>SQLSTTNBR</i> (on page 870) <i>SRVLCNT</i> (on page 875) <i>SRVPTY</i> (on page 883) <i>SVCERRNO</i> (on page 910) <i>SVRCOD</i> (on page 911) <i>TCPPORTHOST</i> (on page 1006) <i>TIMEOUT</i> (on page 1019) <i>TYPESQLDA</i> (on page 1034) <i>UOWID</i> (on page 1038) <i>XIDCNT</i> (on page 1111)
<b>Semantic</b>	<i>ATTLST</i> (on page 105) <i>BIN</i> (on page 118)

*DTAOVR* (on page 321)  
*INHERITANCE* (on page 437)  
*QRYINSID* (on page 688)



**NAME**

BITDR — A Single Bit  
 Name of term prior to Level 2: BIT

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD  
**Codepoint** X'0004'  
**Length** \*  
**Class** CLASS  
**Sprcls** FIELD - A Discrete Unit of Data

A Single Bit (BITDR) is a data value encoding only one of two possible states. The bit is the lowest level of data representation in DDM.

**Length Specification**

The length attribute for BITDR has a value of 1 (bit).

**Literal Form**

Literals are specified in the form B'X' where X is 0 or 1.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
value	ENUVAL	b'0'
	ENUVAL	b'1'
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	
<b>vldattls</b>	NIL	

**SEE ALSO**

**clsvar** *ENUVAL* (on page 347)  
**insvar** *BITSTRDR* (on page 124)  
*CNSVAL* (on page 221)  
*DFTVAL* (on page 276)  
*DSSFMT* (on page 318)  
*MAXVAL* (on page 499)  
*MINVAL* (on page 519)  
*RESERVED* (on page 762)  
*RSLSETFLG* (on page 783)  
*SPCVAL* (on page 831)  
**Semantic** *INHERITANCE* (on page 437)  
*LENGTH* (on page 472)  
*MAXLEN* (on page 495)  
*MINLEN* (on page 517)

**NAME**

BITSTRDR — Bit String Field  
 Name of term prior to Level 2: BITSTR

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD  
**Codepoint** X'0005'  
**Length** \*  
**Class** CLASS  
**Sprcls** FIELD - A Discrete Unit of Data

Bit String Field (BITSTRDR) is an ordered and contiguous collection of bits.

**Length Specification**

The length of a BITSTRDR field is expressed in bits.

**Literal Form**

Literals are specified in the form B'x...', where x is 0 or 1.

<b>clsvar</b>	NIL
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
value	INSTANCE_OF BITDR - A Single Bit REPEATABLE
<b>clscmd</b>	NIL
<b>inscmd</b>	NIL
<b>vldattls</b>	NIL

**SEE ALSO**

**clsvar** *ENUVAL* (on page 347)  
**insvar** *BINDR* (on page 120)  
*BYTDR* (on page 145)  
*CNSVAL* (on page 221)  
*DFTVAL* (on page 276)  
*HEXDR* (on page 428)  
*MAXVAL* (on page 499)  
*MINVAL* (on page 519)  
*RESERVED* (on page 762)  
*SPCVAL* (on page 831)  
**Semantic** *INHERITANCE* (on page 437)  
*LENGTH* (on page 472)  
*MAXLEN* (on page 495)  
*MINLEN* (on page 517)  
*RSLSETFLG* (on page 783)

**NAME**

BNDCHKEXS — Bind Existence Checking

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'211B'  
**Length** \*  
**Class** CLASS  
**Sprcls** STRING - String

Bind Existence Checking (BNDCHKEXS) String controls whether the relational database (RDB) treats the absence of a named RDB object (table, view, and so on) on an SQL statement or the requester not being authorized to a named RDB object as an error. If the RDB treats the absence or the lack of authorization to a named RDB object as an error and the BGNBND command is being executed, then the package may or may not be created depending on the value specified for the Bind Package Creation Control (BNDCRTCTL) parameter. If the RDB treats the absence or the lack of authorization to a named RDB object as an error and the REBIND command is being executed, then the package is not rebound.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	6	
class	X'211B'	
value	INSTANCE_OF	CODPNTDR - Codepoint Data Representation
	ENUVAL	X'241D' - BNDEXSOPT - Bind Object Existence Option
	ENUVAL	X'241C' - BNDEXSQR - Bind Object Existence Required
	DFTVAL	X'241D' - BNDEXSOPT - Bind Object Existence Option
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *BGNBND* (on page 110)  
*REBIND* (on page 753)  
**Semantic** *ENDBND* (on page 336)

**NAME**

BNDCHKONL — Bind Check Only

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2421'**Length** \***Class** codpnt

Bind Check Only (BNDCHKONL) specifies that the target relational database (RDB) performs all syntax and semantic checks on the SQL statements bound to the package. However, a package must not be created as a part of this bind process. This also specifies that if PKGRPLOPT(PKGRPLALW) is specified and an existing package with the same package and version name already exists, then the existing package must not be either dropped or replaced as a part of this bind process.

**SEE ALSO****insvar** *BNDCRTCTL* (on page 127)**Semantic** *BNDNERALW* (on page 132)

**NAME**

BNDCRTCTL — Bind Package Creation Control

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'211D'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Bind Package Creation Control (BNDCRTCTL) String specifies the conditions that govern creating a package with the bind process. This parameter does not apply if the BGNBND command does not successfully initiate the bind process.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	6	
class	X'211D'	
value	INSTANCE_OF	CODPNTDR - Codepoint Data Representation
	ENUVAL	X'2421' - BNDCHKONL - Bind Check Only
	ENUVAL	X'2422' - BNDNERALW - Bind No Errors Allowed
	ENUVAL	X'2423' - BNDERRALW - Bind Errors Allowed
	DFTVAL	X'2422' - BNDNERALW - Bind No Errors Allowed
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *BGNBND* (on page 110)

**Semantic** *BNDCHKEXS* (on page 125)  
*BNDEXSQR* (on page 131)  
*BNDNERALW* (on page 132)  
*PKGOWNID* (on page 600)  
*PKGRPLALW* (on page 602)

**NAME**

BNDERRALW — Bind Errors Allowed

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2423'**Length** \***Class** codpnt

Bind Errors Allowed (BNDERRALW) specifies that the target relational database (RDB) performs all syntax and semantic checks on the SQL statements being bound to the package. Even if errors are detected during the bind process, a package must be created. Reserved sections must be generated in the package for SQL statements found to be in error.

**SEE ALSO****insvar** *BNDCRTCTL* (on page 127)

**NAME**

BNDEXPOPT — Bind Explain Option

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2130'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Bind Explain Option (BNDEXPOPT) String controls whether the target SQLAM causes the target relational database (RDB) to produce explanatory information for all static and dynamic explainable SQL statements in the package. An explainable SQL statement is any statement that begins with SELECT, INSERT, UPDATE, or DELETE.

Explanatory information that the target RDB creates is produced and stored in the normal target RDB manner. The explanatory information is not returned to the source SQLAM during the bind or rebind process.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	6	
class	X'2130'	
value	INSTANCE_OF	CODPNTDR - Codepoint Data Representation
	ENUVAL	X'243B' - EXPALL - Explain All Explainable SQL Statements
	ENUVAL	X'240A' - EXPYES - Explain All Explainable Static SQL Statements
	MINLVL	7
	ENUVAL	X'243A' - EXPNON - Explain No SQL Statements
	DFTVAL	X'243A' - EXPNON - Explain No SQL Statements
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *BGNBND* (on page 110)  
*CODPNTDR* (on page 235)  
*EXPALL* (on page 393)  
*EXPNON* (on page 394)  
*EXPYES* (on page 396)  
*REBIND* (on page 753)

**Semantic** *CLASS* (on page 158)  
*ENDBND* (on page 336)  
*QDDRDBD* (on page 657)  
*SQLAM* (on page 847)  
*STRING* (on page 888)

**NAME**

BINDEXSOPT — Bind Object Existence Option

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'241D'

**Length** \*

**Class** codpnt

Bind Object Existence Option (BINDEXSOPT) specifies that if a named relational database (RDB) object (table, view, and so on) does not exist or the requester is not authorized to a named RDB object, then the RDB must not treat either of those conditions as an error during the bind or rebind process.

This option allows dynamic binding to occur at execution time for some statements that have additional overhead and reduced runtime performance.

**SEE ALSO**

**insvar** *BNDCHKEXS* (on page 125)



**NAME**

BNDEXSRQR — Bind Object Existence Required

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'241C'**Length** \***Class** codpnt

Bind Object Existence Required (BNDEXSRQR) specifies that if a named relational database (RDB) object (table view, and so on) does not exist or the requester is not authorized to a named RDB object, then the RDB must treat this condition as an error during the bind process. For a BGNBND command, this might prevent creation or replacement of the package depending on the value specified for the BNDCRTCTL parameter. For a REBIND command, this might prevent replacement of the package.

**SEE ALSO****insvar** *BNDCHKEXS* (on page 125)

## NAME

BNDNERALW — Bind No Errors Allowed

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD

**Codepoint** X'2422'

**Length** \*

**Class** codpnt

Bind No Errors Allowed (BNDNERALW) specifies that the target RDB performs all syntax and semantic checks on the SQL statements bound to the package. If, however, any error is detected during the bind process, no package is created (or replaced) as a part of the bind process. An error is considered to have been detected:

- When a BNDSQLSTT command returns a DDM reply message and the reply message has a severity code value of ERROR or greater
- When a BNDSQLSTT command that contains an error indication that the SQL language defines returns an SQLCARD reply data object

If an error is detected for a section and the next BNDSQLSTT command reuses that section, errors previously detected for that section are ignored. Only the errors detected by the immediately preceding BNDSQLSTT command detects can be ignored.

If an error is detected, the bind process proceeds as if BNDCHKONL had been specified for the BNDCRTCTL parameter.

If no errors are detected, the package is created or replaced as part of the bind process.

## SEE ALSO

**insvar** *BNDCRTCTL* (on page 127)

**NAME**

BNDOPT — Bind Option

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2405'

**Length** \*

**Class** CLASS

**Sprcls** COLLECTION - Collection Object

BNDOPT is a DDM collection that contains a name of a bind option keyword in BNDOPTNM and the keyword value in BNDOPTVL.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'2405'	
nameof	INSTANCE_OF REQUIRED	BNDOPTNM - Bind Option Name
value	INSTANCE_OF REQUIRED	BNDOPTVL - Bind Option Value
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**cmddda** *BGNBND* (on page 110)  
*REBIND* (on page 753)

**insvar** *BGNBND* (on page 110)

**Semantic** *BGNBND* (on page 110)  
*REBIND* (on page 753)

**NAME**

BNDOPTNM — Bind Option Name

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2144'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

BNDOPTNM (Bind Option Name) specifies the value for a bind keyword name. The value is a character string in the CCSID of the CCSIDMGR negotiated at EXCSAT (or 500 if CCSIDMGR is not supported). Only one keyword name is allowed per occurrence of BNDOPTNM.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
<b>length</b>	*	
<b>value</b>	INSTANCE_OF	CHRSTRDR - Character String Data Representation
	MINLEN	1
	MAXLEN	255
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *BNDOPT* (on page 133)

**Semantic** *BNDOPT* (on page 133)  
*BNDOPTVL* (on page 135)

**NAME**

BNDOPTVL — Bind Option Value

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2145'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

BNDOPTVL (Bind Option Value) specifies the value for the corresponding BNDOPTNM (Bind Option Name) keyword. The value is a character string in the CCSID of the CCSIDMGR negotiated at EXCSAT (or 500 if CCSIDMGR is not supported). Multiple values can be sent in BNDOPTVL in which case the values are separated by hex char FF.

<value1>0xFF<value2>0xFF...0xFF0 (valuen)

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
<b>length</b>	*	
<b>value</b>	INSTANCE_OF	CHRSTRDR - Character String Data Representation
	MINLEN	0
	MAXLEN	32,767
	MINLVL	7
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *BNDOPT* (on page 133)

**Semantic** *BNDOPT* (on page 133)

## NAME

BNDSQLSTT — Bind SQL Statement to an RDB Package

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD

**Codepoint** X'2004'

**Length** \*

**Class** CLASS

**Sprcls** COMMAND - Command

Bind SQL Statement to an RDB Package (BNDSQLSTT) Command binds an SQL statement and any referenced application variable definitions to a relational database (RDB) package.

**Source System Processing**

The source system determines the location of the RDB:

Local Call the local RDB server.

Remote Send the BNDSQLSTT command to the remote RDB server.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**DSS Carrier: RQSDSS****Target System Processing**

The *bdsttasm* parameter specifies the assumptions that the source server program preparation process (the precompiler) made about the SQL statement.

The *pkgnamcsn* parameter specifies the fully qualified package name, its consistency token, and a section number. This parameter identifies the package in which the SQL statement can be bound. The fully qualified package name and consistency token portions of the *pkgnamcsn* parameter must be the same as specified by the *pkgnamct* parameter of the BGNBND command that started the package binding process. The *pkgnamcsn* parameter also specifies the number of the package section being used for the SQL statement.

The *rdbnam* parameter specifies the name of the RDB that the ACCRDB command accesses. If the *rdbnam* parameter is specified, its value must be the same as the value specified on the ACCRDB command for *rdbnam*.

The *sqlsttnbr* parameter specifies the source application statement number of the SQL statement that this command is binding.

The TYPDEFNAM command data object specifies the name of the data type to data representation mapping definitions that the source SQLAM uses when sending command data objects for this command.

The TYPDEFOVR command data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type to data representation mapping definitions for the command data objects for this command.

The BNDSQLSTT command must be followed by an SQLSTT command data object that contains the SQL statement being bound into the package and optionally by an SQLSTTVRB command data object that describes the variables that the SQL statement references. An SQLSTTVRB must be sent if the SQL statement contains variables. The SQLSTT must be sent prior to the SQLSTTVRB if an SQLSTTVRB is sent. The SQLSTT FD:OCA descriptor describes the contents of the SQLSTT command data object. The SQLSTTVRB FD:OCA descriptor describes the contents of the SQLSTTVRB command data object.

The SECTKNOVR command data object is sent by the intermediate server, if the SQLSTT and the optional SQLSTTVRB objects are encrypted. The SECTKNOVR contains the encryption seed and the encryption token used to encrypt the SQLSTT and optional SQLSTTVRB command data objects. The SECTKNOVR DSS is encrypted and must be sent prior to the encrypted SQLSTT command data object.

The TYPDEFNAM reply data object specifies the name of the data type to data representation mapping definitions that the target SQLAM uses when sending the SQLCARD object for this command.

The TYPDEFOVR reply data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type to data representation mapping definitions for the SQLCARD object for this command.

The BNDSQLSTT command might only be specified as part of the package binding process after a BGNBND command and before an explicit or implicit ENDBND command.

If the target RDB detects an error in an SQL statement, the source SQLAM is allowed to use the same section number on the next BNDSQLSTT command for a different SQL statement.

Normal completion of the BNDSQLSTT command returns an SQLCARD object. An RDBUPDRM must be returned if the command makes recoverable updates in the RDB and the server did not return a prior RDBUPDRM in the current unit of work. A recoverable update is an RDB update that writes information to the RDB's recovery log.

### Data Conversion

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

### Exceptions

Exception conditions the RDB detects are reported in an SQLCARD object.

If the package binding process is not active, then the command is rejected with the PKGBNARM.

If the fully qualified package name and consistency token portions of the *pkgnamcsn* parameter are not the same as the *pkgnamct* parameter of the BGNBND command that started the current package binding process specifies, then the command is rejected with the PKGBNARM.

If the target SQLAM detects that the values of the SQLSTT or SQLSTTVRB command data objects do not have the characteristics that the FD:OCA descriptor claims, then the command is rejected with the DTAMCHRM.

If the specified RDB is not currently accessed, then the command is rejected with the RDBNACRM.

If an interrupt RDB request (INTRDBRQS) command terminates the command, then the SQLERRRM is returned.

**Source System Reply Processing**

Return any reply messages and data objects to the requester.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**Package Name and Consistency Token**

With SQLAM level 7, the PKGNAMCSN is changed to an optional instance variable. If specified, the values are used as the default package name and consistency token for the connection. If not specified on the command, the PKGSN is required to identify the section number. In this case, the default package name, consistency token and specified section number is used for the request. If PKGNAMCSN was never specified on the connection, a PRCCNVRM is returned.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>		<b>CLASS INSTANCE VARIABLES</b>
length	*	
class	X'2004'	
bndsttasm	INSTANCE_OF OPTIONAL	BNDSTTASM - Bind SQL Statement Assumptions
pkgnamcsn	INSTANCE_OF MINLVL OPTIONAL MTLEXC	PKGNAMCSN - RDB Package Name, Consistency Token, and Section Number 7 Required if PKGSN not specified PKGSN
pkgsn	INSTANCE_OF MINLVL OPTIONAL MTLEXC	PKGSN - RDB Package Section Number 7 Required if PKGNAMCSN not specified. PKGNAMCSN
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
sqlsttnbr	INSTANCE_OF OPTIONAL	SQLSTTNBR - SQL Statement Number
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>cmddta</b>		<b>COMMAND OBJECTS</b>
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL MINLVL ENUVAL	TYPDEFNAM - Data Type Definition Name 'QTDSQL370' 'QTDSQL400' 'QTDSQLX86' 'QTDSQLASC' 4 'QTDSQLVAX'



	MINLVL	4
	OPTIONAL	
	DFTVAL	''
	NOTE	The default means the value received on ACCRDB command is used.
	REPEATABLE	
X'0035'	INSTANCE_OF	TYPDEFOVR - TYPDEF Overrides
	OPTIONAL	
	DFTVAL	''
	NOTE	The default means the value received on ACCRDB command is used.
	REPEATABLE	
X'190B'	INSTANCE_OF	SECTKNOVR - SECTKN Override
	OPTIONAL	
	NOTE	This is sent by the intermediate server, if the SQLSTT and the optional SQLSTTVRB command data objects are encrypted. This must precede the SQLSTT command data object.
X'2414'	INSTANCE_OF	SQLSTT - SQL Statement
	REQUIRED	
	NOTE	This must precede the SQLSTTVRB command data object.
X'2419'	INSTANCE_OF	SQLSTTVRB - SQL Statement Variable Descriptions
	OPTIONAL	
	NOTE	Describes the host program variables the SQL statement being bound references. This must follow the SQLSTT command data object if the SQL statement contains application variable references.
<b>rpydta</b>	<b>REPLY OBJECTS</b>	
X'002F'	INSTANCE_OF	TYPDEFNAM - Data Type Definition Name
	ENUVAL	'QTDSQL370'
	ENUVAL	'QTDSQL400'
	ENUVAL	'QTDSQLX86'
	ENUVAL	'QTDSQLASC'
	MINLVL	4
	ENUVAL	'QTDSQLVAX'
	MINLVL	4
	OPTIONAL	
	DFTVAL	''
	NOTE	The default means the value received on ACCRDBRM is used.
X'0035'	INSTANCE_OF	TYPDEFOVR - TYPDEF Overrides
	OPTIONAL	
	DFTVAL	''
	NOTE	The default means the value received on ACCRDBRM is used.

X'1C03'	INSTANCE_OF OPTIONAL MINLVL	MGRVLVLOVR - Manager Level Overrides 7
X'2408'	INSTANCE_OF REQUIRED	SQLCARD - SQL Communications Area Reply Data
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'220D'	INSTANCE_OF	ABNUOWRM - Abnormal End Unit of Work Condition
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported
X'220E'	INSTANCE_OF	DTAMCHRM - Data Descriptor Mismatch
X'1253'	INSTANCE_OF	OBJNSPRM - Object Not Supported
X'2206'	INSTANCE_OF	PKGBNARM - RDB Package Binding Not Active
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'2204'	INSTANCE_OF	RDBNACRM - RDB Not Accessed
X'2218'	INSTANCE_OF MINLVL	RDBUPDRM - RDB Update Reply Message 4
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limits Reached
X'2213'	INSTANCE_OF MINLVL NOTE	SQLERRRM - SQL Error Condition 4 The SQLERRRM is only returned when the BNDSQLSTT command is terminated by the processing of an INTRDBRQS command.
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'125F'	INSTANCE_OF	TRGNSPRM - Target Not Supported
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

**SEE ALSO**

<b>inscmd</b>	<i>SQLAM</i> (on page 847)
<b>insvar</b>	<i>MAXSCTNBR</i> (on page 498)
<b>Semantic</b>	<i>BGNBND</i> (on page 110) <i>BNDNERALW</i> (on page 132) <i>ENDBND</i> (on page 336) <i>PKGBNARM</i> (on page 585) <i>REBIND</i> (on page 753) <i>SECTKNOVR</i> (on page 822) <i>SQL</i> (on page 835) <i>SQLAM</i> (on page 847) <i>SQLERRRM</i> (on page 864)

**NAME**

BNDSTTASM — Bind SQL Statement Assumptions

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2126'  
**Length** \*  
**Class** CLASS  
**Sprcls** STRING - String

Bind SQL Statement Assumptions (BNDSTTASM) String specify the assumptions made by the source server program preparation process (the precompiler) makes when it was unable to properly classify the SQL statement contained in the SQLSTT command data object. If the target relational database (RDB) detects an SQL statement that does not match, then the target RDB must reject the SQL statement and return an appropriate SQLCARD reply data object.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	6	
class	X'2126'	
value	INSTANCE_OF	CODPNTDR - Codepoint Data Representation
	ENUVAL	X'2436' - STTSCCCLS - Statement Successfully Classified
	ENUVAL	X'2437' - STTASMEUI - Statement Assumptions Executable Unique Section Input
	DFTVAL	X'2436' - STTSCCCLS - Statement Successfully Classified
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *BNDSQLSTT* (on page 136)

**NAME**

BOOLEAN — Logical Value

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD  
**Codepoint** X'0006'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

Truth State (BOOLEAN) Scalar Object specifies a logical value of TRUE or FALSE.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	5	
class	X'0006'	
value	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	LENGTH	2
	ENUVAL	X'F0' - FALSE - False State
	ENUVAL	X'F1' - TRUE - True State
	REQUIRED	
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**Semantic** *DUPQRYOK* (on page 322)  
*FORGET* (on page 423)  
*INHERITANCE* (on page 437)  
*OUTEXP* (on page 571)  
*QRYATTSET* (on page 666)  
*RDBALWUPD* (on page 726)  
*RDBCMTOK* (on page 731)  
*RLSCONV* (on page 765)  
*RTNSQLDA* (on page 795)  
*SQLCSRHLD* (on page 858)

**NAME**

BUFINSIND — Buffered Insert Indicator

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'215C'  
**Length** \*  
**Class** CLASS  
**Sprcls** STRING - String

Buffered Insert Indicator (BUFINSIND) specifies whether and how the target server should optimize an SQL INSERT statement using the buffered insert technique when it gets executed as an atomic multi-row input operation against a partitioned database. The following options are available:

- Do not use buffered insert optimization.
- Use buffered insert. Any duplicate row error (SQLSTATE 23505) is returned as-is.
- Use buffered insert. Any duplicate row error (SQLSTATE 23505) is downgraded to a warning (SQLSTATE 01661).

The BUFINSIND has no effect and is ignored by the target server unless all of the following conditions are met:

- The SQL statement being prepared is an INSERT.
- The SQL statement is executed as an atomic multi-row input operation.
- The target database is partitioned.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'215C'	
<b>value</b>	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	MINLVL	7
	LENGTH	2
	ENUVAL	X'00'
	NOTE	Do not use buffered insert.
	ENUVAL	X'01'
	NOTE	Use buffered insert. Any duplicated row error (SQLSTATE 29505) is returned as-is.
	ENUVAL	X'02'
	NOTE	Use buffered insert. Any duplicate row error (SQLSTATE 29505) is downgraded to a warning (SQLSTATE 01661).
	IGNORABLE	If the SQL statement is not an INSERT, or if the statement is not executed as an atomic multi-row input operation, or if the target database is not partitioned.

	DFTVAL	X'00'
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar**      *PRPSQLSTT* (on page 636)

**NAME**

BYTDR — An 8-bit Data Representation Value

Name of term prior to Level 2: BYTE

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'0043'

**Length** \*

**Class** CLASS

**Sprcls** FIELD - A Discrete Unit of Data

An 8-bit value (BYTDR) defines each byte of data representation which requires a string containing eight bits.

**Length Specification**

The length of a BYTDR field is always one byte.

**Literal Form**

The literal form of a byte is a two-position hexadecimal string; for instance, 01 or 3D'.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
value	INSTANCE_OF LENGTH	BITSTRDR - Bit String Field 8
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	
<b>vldattls</b>	NIL	

**SEE ALSO**

**insvar** *ATMIND* (on page 104)  
*BYTSTRDR* (on page 146)  
*ENDCHNTYP* (on page 341)  
*RSYNCTYP* (on page 790)  
*SYNCTYPE* (on page 984)  
*UOWSTATE* (on page 1040)

**Semantic** *INHERITANCE* (on page 437)

NAME

BYTSTRDR — A Byte String

DESCRIPTION (Semantic)

**Dictionary** QDDPRMD

**Codepoint** X'0044'

**Length** \*

**Class** CLASS

**Sprcls** FIELD - A Discrete Unit of Data

Byte string (BYTSTRDR). A Discrete Unit of Data is an ordered and contiguous sequence of bytes.

**Length Specification**

The length of a byte string (BYTSTRDR) is specified in bytes.

**Literal Form**

The literal form of a byte string is a hexadecimal string whose length is an even number; for example, X'01010101' or X'3D2B11'.

<b>clsvar</b>	NIL	
<b>insvar</b>		
value	INSTANCE_OF	BYTDR - An 8-bit Data Representation
	REPEATABLE	Value (QDDPRMD)
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	
<b>vldattls</b>	NIL	

SEE ALSO

- clsvar** LOGNAME (on page 484)
- insvar**
  - ARMCORR (on page 100)
  - CNNTKN (on page 220)
  - CRRTKN (on page 246)
  - EXTDTA (on page 397)
  - FDODSC (on page 406)
  - FDODTA (on page 408)
  - FDOEXT (on page 410)
  - FDOOFF (on page 412)
  - MGRNAM (on page 513)
  - OBJDSS (on page 536)
  - OUTOVR (on page 572)
  - PKGCNSTKN (on page 588)
  - PKTOBJ (on page 608)
  - PLGINID (on page 611)
  - PRDDTA (on page 630)
  - PRDID (on page 631)
  - QRYDSC (on page 685)
  - QRYDTA (on page 686)



*RSNCOD* (on page 786)  
*SECTKN* (on page 820)  
*SPVNAM* (on page 834)  
*SQLATTR* (on page 854)  
*SQLCARD* (on page 855)  
*SQLCINRD* (on page 857)  
*SQLDARD* (on page 859)  
*SQLOBJNAM* (on page 866)  
*SQLRSLRD* (on page 867)  
*SQLSTT* (on page 868)  
*SQLSTTVRB* (on page 871)  
*SRVDGN* (on page 873)  
*SRVRLSLV* (on page 884)  
*UOWID* (on page 1038)

**Semantic**

*SECTKN* (on page 820)

**NAME**

CCSIDDBC — CCSID for Double-byte Characters

**DESCRIPTION (Semantic)****Dictionary** QDDBASD**Codepoint** X'119D'**Length** \***Class** CLASS**Sprcls** STRING - String

CCSID for Double-byte Characters (CCSIDDBC) string specifies a coded character set identifier (CCSID) for double-byte characters.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	6	
class	X'119D'	
ccsid	INSTANCE_OF LENGTH	HEXSTRDR - Hexadecimal String 4
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar**            *PKGDFTC* (on page 589)  
                      *TYPDEFOVR* (on page 1030)

**Semantic**        *TYPDEFOVR* (on page 1030)

**NAME**

CCSIDMBC — CCSID for Mixed-byte Characters

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'119E'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

CCSID for Mixed-byte Characters (CCSIDMBC) String specifies a coded character set identifier (CCSID) for mixed-byte characters.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	6	
class	X'119E'	
ccsid	INSTANCE_OF LENGTH	HEXSTRDR - Hexadecimal String 4
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

- insvar** *PKGDFTC* (on page 589)  
*TYPDEFOVR* (on page 1030)
- Semantic** *TYPDEFOVR* (on page 1030)

## NAME

CCSIDMGR — CCSID Manager

## DESCRIPTION (Semantic)

**Dictionary** QDDBASD**Codepoint** X'14CC'**Length** \***Class** CLASS**Sprcls** MANAGER - Resource Manager

CCSID Manager (CCSIDMGR) Resource Manager provides character data conversion of the DDM parameters containing character data. Servers whose native character encoding scheme is ASCII can send character data in ASCII instead of EBCDIC codepoints.

The initial EXCSAT command negotiates the desired encoding scheme. The source server in the *mgrlvls* parameter includes the codepoint of the CCSIDMGR and the CCSID of the code page it uses to send character data. The target server in the *mgrlvls* parameter returns the codepoint of the CCSIDMGR and the CCSID of the code page it uses to send character data. The receiver of character data in the DDM parameters is responsible for transformation of the character data into the code page it supports.

Prior to DDM Level 4, all character data in the DDM parameters had to be in CCSID 500. The CCSIDMGR must support at a minimum the CCSIDs shown in Table 3-2.

**Table 3-2** Required CCSID Support for the CCSIDMGR

Encoding	CCSID	Name	Date
EBCDIC	500	International Latin-1	1986
ASCII	819	ISO/ANSI Multilingual	1987
ASCII	850	Personal Computer Multilingual	1986

**Rules for the CCSIDMGR**

If the CCSIDMGR is not supported in the source server, the source server only supports CCSID 500. It does not support any conversion or transformation of the DDM parameters. The CCSIDMGR is not included in the *mgrlvls* parameter of the EXCSAT command.

If the CCSIDMGR is not supported in the target server, the target server only supports CCSID 500. It does not support any conversion or transformation of the DDM parameters containing character data.

If the CCSIDMGR is supported in the source server, the *mgrlvls* parameter contains the codepoint of the CCSIDMGR, and the preferred CCSID of the code page the source server will use for the DDM parameters containing character data:

- If the CCSID the source server specifies is 500, 819, or 850, then the target server must respond with a CCSID of 500, 819, or 850.
- If the CCSID the source server specifies is not 500, 819, or 850, then the target server can respond with a valid CCSID.
- If the target server does not support the CCSIDMGR, the *mgrlvls* parameter returned for the EXCSAT command contains the codepoint of the CCSIDMGR and the value zero (0) for the level number. The source server must use CCSID 500 for the DDM parameters containing

character data.

- If the target server supports the CCSIDMGR, but not the CCSID requested in the EXCSAT *mgrlvls* parameter, then the target server returns the value FFFF in the *mgrlvls* parameter. The source server must return another EXCSAT command and specify one of the required CCSIDs.
- If the target server supports the CCSIDMGR and the CCSID requested, it returns the target's preferred CCSID of the code page for the DDM parameters containing character data. If the source server does not support the returned CCSID, the source server can send another EXCSAT specifying one of the required CCSIDs (500, 819, or 850) and the target must respond with one of the required CCSIDs (500, 819, 850), or the source server can deallocate the conversation.
- If both servers support the CCSIDMGR and the CCSIDs specified, each server sends parameters containing character data in the appropriate CCSID.

**EXAMPLES**

An example of the CCSIDMGR's effect and its ease of use is shown with the DCLFIL command. Assume the EXCSAT command processing has completed successfully. The source server sends character data in CCSID 819 (ASCII), and the target server uses CCSID 500 (EBCDIC).

```
DCLFIL (DCLNAM(d001) FILNAM(ABCD))
```

as a hexadecimal string:

```
0014 102C 0008 1136 6430 3031 0008 110E 4142 4344
```

The DCLNAM remains 64303031 (d001).

The FILNAM is converted by the target server to C1C2C3C4 (ABCD).

If the FILNAM is returned in a reply message, it is coded as:

```
0008 110E C1C2 C3C4
```

which is converted by the source server to:

```
4142 4344 (ABCD)
```

The EXCSAT command parameter of *extnam* is in the CCSID of the source server. The EXCSATRD parameter of *extnam* is in the CCSID of the target server. Thus:

source sends

```
EXTNAM(PAYROLL5) == 000C 115E 5041 5952 4F4C 4C35
```

target sends

```
EXTNAM(Rm112543) == 000C 115E D994 F1F1 F2F5 F4F3
```

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
<b>length</b>	<b>*</b>
<b>class</b>	<b>X'14CC'</b>
<b>clscmd</b>	<b>NIL</b>

<b>inscmd</b>	<b>NIL</b>	
<b>mgrlvln</b>	<b>REDEFINED</b>	
<b>mgrlvln</b>	INSTANCE_OF NOTE	CCSID - Coded Character Set Identifier The CCSID is an UNSBINDR number. CCSIDs 500, 819, and 850 must be supported. Other CCSIDs are optional.
	SPCVAL NOTE	0 When sent by the target server, zero means the CCSIDMGR is not supported.
	SPCVAL NOTE	65535 When sent by the target server, this value means the CCSID the source server requested is not supported. 65535 is FFFF.
<b>mgrdepls</b>	<b>NIL</b>	
<b>vldattls</b>	<b>VALID ATTRIBUTES</b>	
X'0019'	INSTANCE_OF	HELP - Help Text
X'1452'	INSTANCE_OF	MGRNAM - Manager Name
X'0045'	INSTANCE_OF	TITLE - Title

**SEE ALSO**

<b>insvar</b>	<i>MGRLVL</i> (on page 506)
<b>mgrdepls</b>	<i>AGENT</i> (on page 61)
<b>Semantic</b>	<i>ACCRDB</i> (on page 42) <i>ACCSEC</i> (on page 52) <i>AGENT</i> (on page 61) <i>BGNBND</i> (on page 110) <i>BNDOPTNM</i> (on page 134) <i>BNDOPTVL</i> (on page 135) <i>BNDSQLSTT</i> (on page 136) <i>CHRDR</i> (on page 155) <i>CLSQR</i> Y (on page 165) <i>CNTQR</i> Y (on page 222) <i>DRPPKG</i> (on page 293) <i>DSCRDBTBL</i> (on page 300) <i>DSCSQLSTT</i> (on page 304) <i>ENDBND</i> (on page 336) <i>EXCSAT</i> (on page 363) <i>EXCSQLIMM</i> (on page 371) <i>EXCSQLSET</i> (on page 377) <i>EXCSQLSTT</i> (on page 381) <i>EXTENSIONS</i> (on page 400) <i>KERSECPPL</i> (on page 470) <i>MGROVR</i> (on page 514) <i>OPNQRY</i> (on page 555) <i>PLGINNM</i> (on page 614) <i>PLGINPPL</i> (on page 619) <i>PRPSQLSTT</i> (on page 636) <i>RDBCMM</i> (on page 728) <i>RDBRLLBCK</i> (on page 745)

*REBIND* (on page 753)  
*SECCHK* (on page 800)

**NAME**

CCSIDSBC — CCSID for Single-byte Characters

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'119C'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

CCSID for Single-byte Characters (CCSIDSBC) String specifies a coded character set identifier (CCSID) for single-byte characters.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	6	
class	X'119C'	
ccsid	INSTANCE_OF LENGTH	HEXSTRDR - Hexadecimal String 4
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

- insvar** *PKGDFTC* (on page 589)  
*TYPDEFOVR* (on page 1030)
- Semantic** *ACCRDB* (on page 42)  
*TYPDEFOVR* (on page 1030)



**NAME**

CHRDR — A Graphic Character

**DESCRIPTION (Semantic)****Dictionary** QDDPRMD**Codepoint** X'0008'**Length** \***Class** CLASS**Sprcls** FIELD - A Discrete Unit of Data

A Graphic Character (CHRDR) Encoded Information includes the graphic characters listed in Table 3-3. The additional code pages for ASCII allow servers to send in the DDM commands and reply message parameters containing character data in either EBCDIC or ASCII code pages. See *CCSIDMGR* (on page 150).

The CCSIDMGR must support the code pages shown in Table 3-3; see the *Character Data Representation Architecture Reference* (SC09-1390, IBM).

**Table 3-3** Required CCSIDs

Encoding	CCSID	Name	Date
EBCDIC	500	International Latin-1	1986
ASCII	819	ISO/ANSI Multilingual	1987
ASCII	850	Personal Computer Multilingual	1986

**clsvar** NIL**insvar** NIL**clscmd** NIL**inscmd** NIL**vldattls** NIL**SEE ALSO**

**insvar** *CHRSTRDR* (on page 156)  
*NAMSYMDR* (on page 528)

**Semantic** *DTAOVR* (on page 321)  
*INHERITANCE* (on page 437)  
*LENGTH* (on page 472)  
*MAXLEN* (on page 495)  
*MINLEN* (on page 517)

**NAME**

CHRSTRDR — Character String  
 Name of term prior to Level 2: CHRSTR

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD  
**Codepoint** X'0009'  
**Length** \*  
**Class** CLASS  
**Sprcls** FIELD - A Discrete Unit of Data

Character String (CHRSTRDR) is an ordered and contiguous collection of characters.

**Length Specification**

The length of a CHRSTRDR field is expressed in bytes.

**Literal Form**

Character string literals are represented by quoted strings; for instance:

"bcde" or "Now is the time ..."

When a quote character is required in the string, specify two single quotes, as in the string:

'DDM's approach to architecture documentation...'

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
value	INSTANCE_OF	CHRDR - A Graphic Character
	REPEATABLE	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	
<b>vldattls</b>	NIL	

**SEE ALSO**

**insvar** *BNDOPTNM* (on page 134)  
*BNDOPTVL* (on page 135)  
*DFTRDBCOL* (on page 274)  
*LOGTSTMP* (on page 485)  
*NAMDR* (on page 526)  
*NAME* (on page 527)  
*NEWPASSWORD* (on page 531)  
*NOTE* (on page 533)  
*PASSWORD* (on page 578)  
*PKGID* (on page 591)  
*PKGNAME* (on page 594)  
*PKGNAMECSN* (on page 596)  
*PKGNAMECT* (on page 598)  
*PKGOWNID* (on page 600)  
*PLGINNM* (on page 614)

	<i>PLGINPPL</i> (on page 619)
	<i>RDBCOLID</i> (on page 732)
	<i>RDBNAM</i> (on page 736)
	<i>RSCNAM</i> (on page 781)
	<i>SNAADDR</i> (on page 827)
	<i>SRVCLSNM</i> (on page 872)
	<i>TCPHOST</i> (on page 999)
	<i>TEXT</i> (on page 1018)
	<i>TITLE</i> (on page 1020)
	<i>UOWID</i> (on page 1038)
	<i>USRID</i> (on page 1044)
<b>Semantic</b>	<i>INHERITANCE</i> (on page 437)
	<i>KERSECPPL</i> (on page 470)
	<i>LENGTH</i> (on page 472)
	<i>MAXLEN</i> (on page 495)
	<i>MINLEN</i> (on page 517)
	<i>RDBCOLID</i> (on page 732)
<b>semantic</b>	<i>CLASS</i> (on page 158)

## NAME

CLASS — Object Descriptor

## DESCRIPTION (Semantic)

**Dictionary** QDDPRMD**Codepoint** X'000A'**Length** \***Class** CLASS**Sprcls** COLLECTION - Collection Object

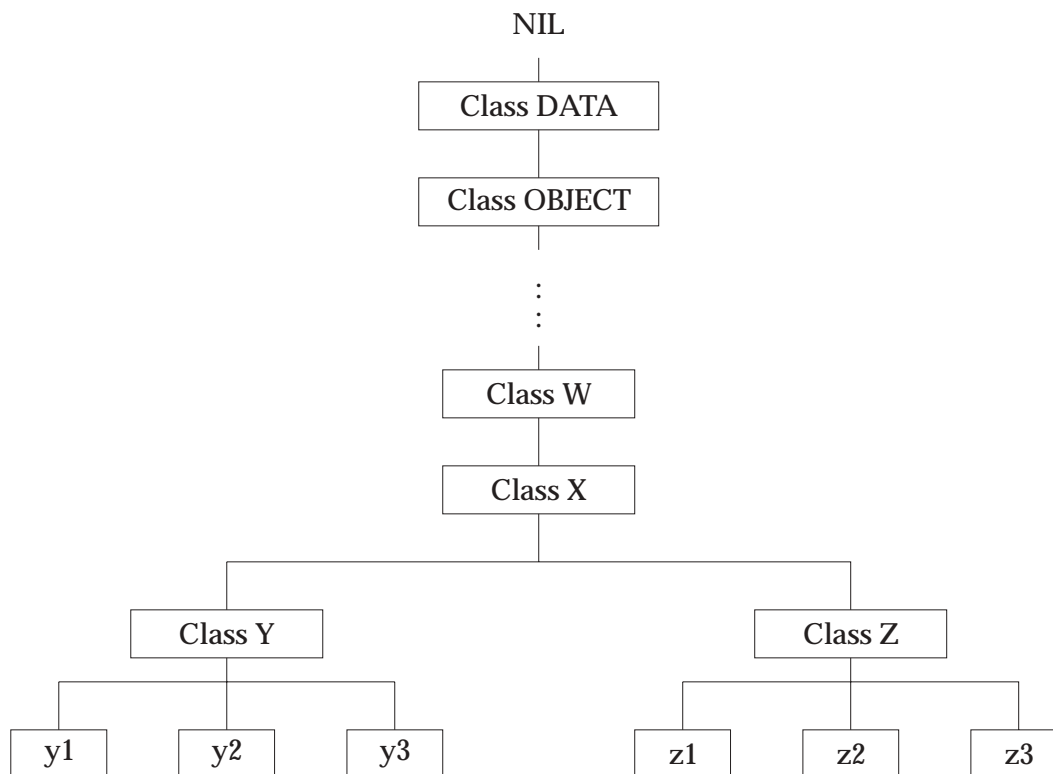
Object Descriptor (CLASS) Collection Object is an object that describes a set of objects that have a common structure and respond to the same commands. These objects are called the instances of the class.

The INSTANCE VARIABLE list describes the internal memory structure of an object. Structurally, the classes describe three general categories of objects with variables as follows:

1. Primitive data
  - No preceding length or class variables
  - Variables consist of one or more concatenated data fields
  - Example: RQSDSS
2. Scalar objects
  - Includes preceding length and class variables
  - Variables consist of one or more concatenated data fields
  - Example: DUPFILOP
3. Collection objects
  - Include preceding length and class variables for the collection as a whole
  - Variables consist of one or more scalar or collection objects, each with its own length, class, and other variables
  - Examples: CRTSEQF command and the ENDFILRM message

**Class Hierarchies**

Classes are arranged in a hierarchy where descriptions of structure (named variables) and supported commands are inherited. In each class, the superclass (**sprcls**) variable names its inherited class. This hierarchy is illustrated in Figure 3-12 (on page 159). See *INHERITANCE* (on page 437), *INHERITED* (on page 442), and *SPRCLS* (on page 832). The superclass hierarchy defines a superclass chain from the class's immediate superclass to DATA.



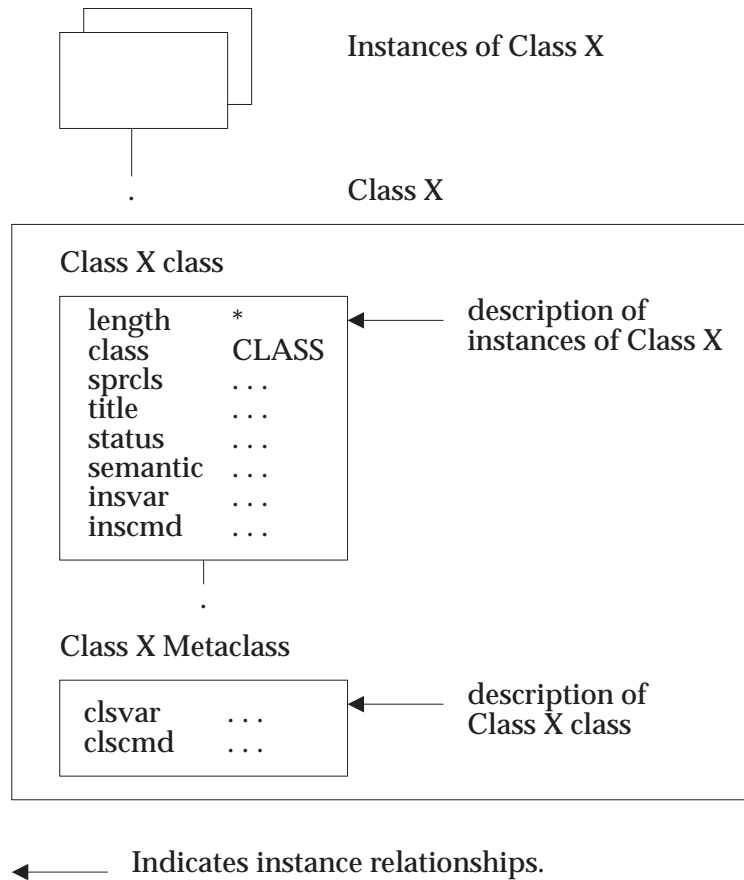
**Figure 3-12** Superclass, Class, and Instance Relationships

#### Figure Notes

- DATA has no superclass.
- OBJECT is the superclass of classes W, X, Y, and Z.
- The superclass chain of both Y and Z is X, W, ... OBJECT, DATA.
- y1, y2, and y3 are instances of Class Y.
- z1, z2, and z3 are instances of Class Z.

#### Metaclasses

Class objects provide descriptive information about their instances. Classes also describe themselves. Each class is an instance of a metaclass that the **clsvar** and **clscmd** variables of the class describe. Thus, the variables within a class are described by their own **clsvar** variable, and the commands the class supports are specified in their own **clscmd** variable. Metaclasses, which are an informal concept in DDM, are illustrated in Figure 3-13 (on page 160).



**Figure 3-13** Class to Metaclass Relationships

**Figure Note**

Classes and metaclasses are combined into a single CLASS object in DDM with a single set of variables.

The inheritance of variables and commands from a class's superclass also applies to the **clsvar** and **clscmd** metaclass variables. However, the superclass chain for metaclass variables extends from class OBJECT to class CLASS and is terminated. Thus, class CLASS (the term being read) is fully self-describing, and all other classes inherit from it through their superclass chain. This is illustrated in Figure 3-14 (on page 161).

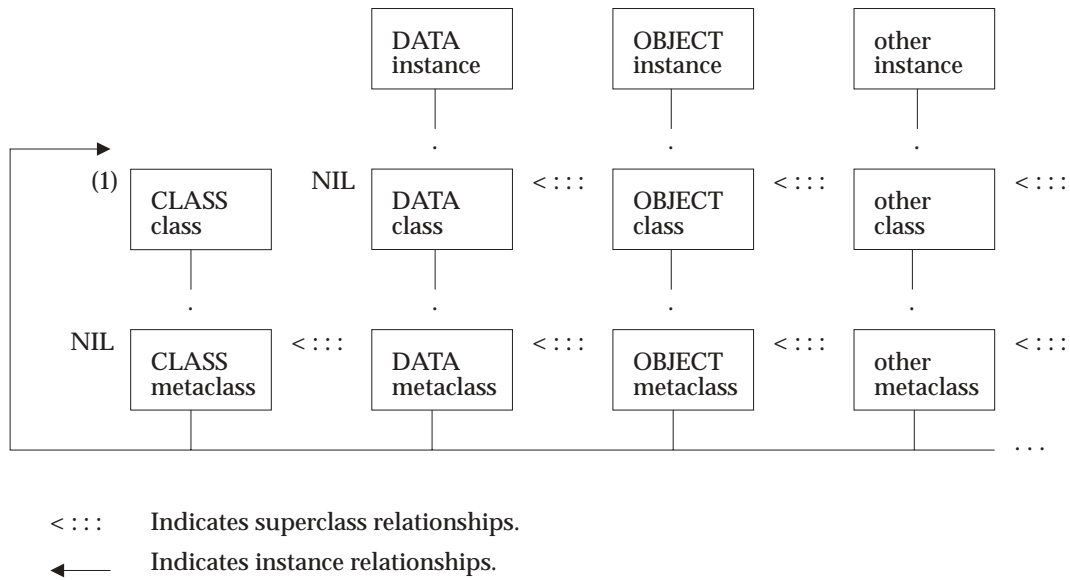


Figure 3-14 Superclass and Super-Metaclass Chains

Figure Notes

- Classes and metaclasses are combined as a single CLASS object in DDM.
- NIL is the end of a superclass chain.
- The superclass of CLASS class loops back through the OBJECT class.

Mapping Objects into Memory

All DDM terms are designed as objects and can therefore be mapped into memory or transmitted as objects. The documentation of individual DDM terms, however, suppresses redundant detail information. For example, most terms have a **semantic** variable that is defined as the class HELP. The value of the **semantic** variable is presented as formatted text without the length, class, and title variables of a complete instance of HELP. The values of these variables should be defaulted as required by their class or assumed to be NIL. Figure 3-15 (on page 162) shows how a class instance would be mapped into memory as a collection of objects.

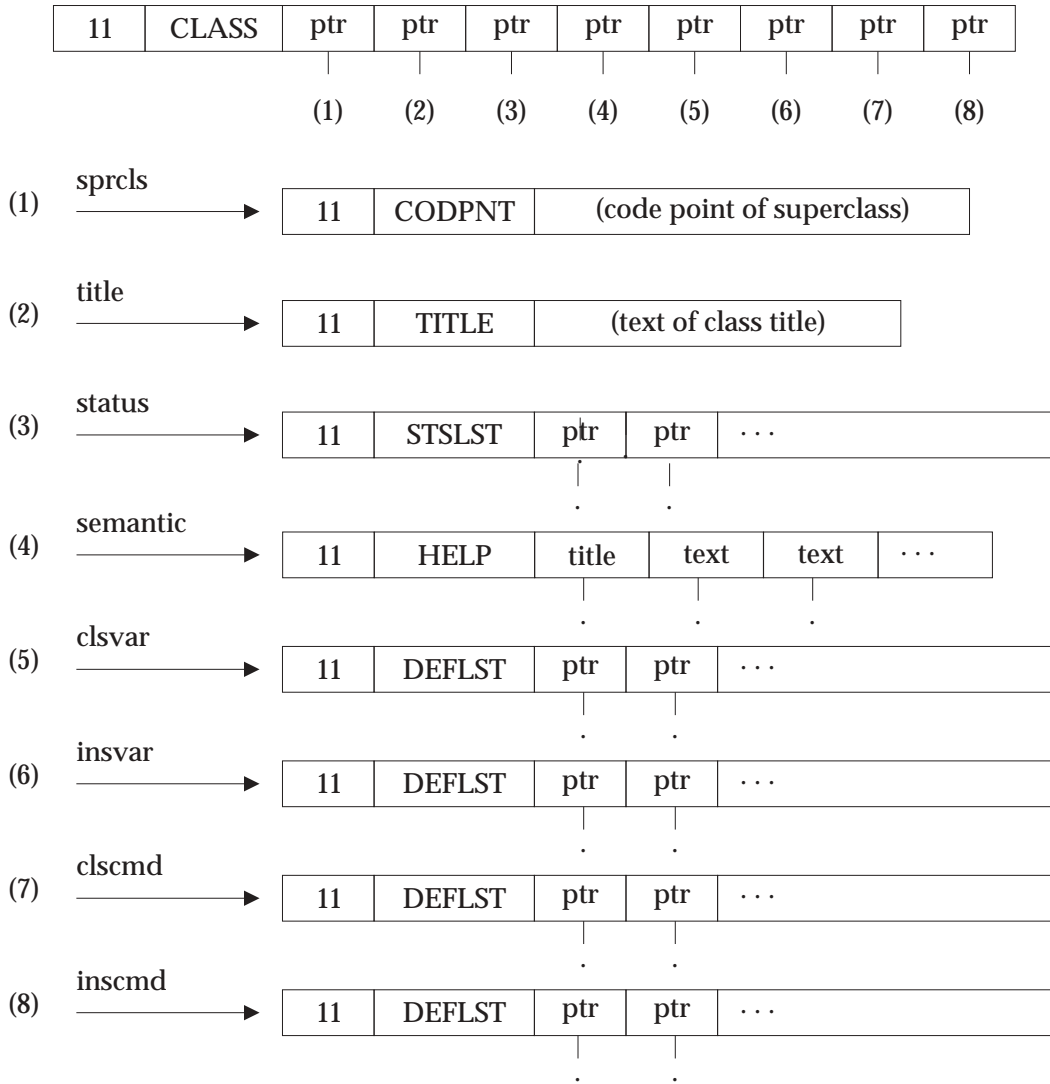


Figure 3-15 Class Structure Overview

An ellipsis (...) denotes repeated variables.

**Figure Notes**

- Each structure is an object with the name of its class and selected variables specified.
- Arrows denote references to independent objects stored in the dictionary.

clsvar	CLASS VARIABLES	
length	INSTANCE_OF LENGTH	BINDR - Binary Number Field 16



	NOTE	The length (in bytes) of an object includes its own length and the length of all subsequent variables in the object.
<b>class</b>	INSTANCE_OF NOTE	CODPNTDR - Codepoint Data Representation Specifies the codepoint of the class of the object.
<b>sprcls</b>	INSTANCE_OF NOTE	CODPNT - Codepoint Specifies the codepoint of the superclass of the class being defined. The new class inherits variables and commands from its superclass. See the description of SPRCLS, INHERITANCE, and INHERITED.
<b>status</b>	INSTANCE_OF TITLE NOTE	STSLST - Term Status Collection term status Specifies the status of the class.
<b>title</b>	INSTANCE_OF NOTE	TITLE - Title The title of a class is a brief descriptive phrase that can be presented wherever the architecture specification references the class by name.
<b>semantic</b>	INSTANCE_OF TITLE NOTE	CHRSTRDR - Character String Data Representation description The semantic text of a class describes the class and provides information common to all instances of the class.
<b>clsvar</b>	INSTANCE_OF TITLE INHERITED NOTE	DEFLST - Definition List class variables  Defines only the variables of the class, not the class's instance variables. These variables represent the structure of the class, they contain information common to all instances of the class. Class variables are addressable through the class object, not through the instances of the class. Additional class variables can be defined for each class. Each definition in the list describes a single named class variable.
<b>insvar</b>	INSTANCE_OF TITLE INHERITED NOTE	DEFLST - Definition List instance variables  Defines the variables that apply only to an instance of the class. Each definition in the list describes a single named instance variable.
<b>clscmd</b>	INSTANCE_OF TITLE INHERITED NOTE	DEFLST - Definition List class commands  Defines a set of commands describing the operations that can be performed by the specified class; typically, these are instance creation and initialization. Each definition in the list describes a

		single class command whose name is specified by the codepoint of its class.
<b>inscmd</b>	INSTANCE_OF TITLE INHERITED NOTE	DEFLST - Definition List instance commands  Defines a set of commands that can be performed by instances of the class. Each definition in the list describes a single instance command whose name is specified by the codepoint of its class.
<b>insvar</b>		See <b>clsvar</b> above.
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

None.

## NAME

CLSQR — Close Query

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD

**Codepoint** X'2005'

**Length** \*

**Class** CLASS

**Sprcls** COMMAND - Command

Close Query (CLSQR) command closes a query that an OPNQR command opened.

**Source System Processing**

The source system determines the location of the RDB:

Local Call the local RDB server.

Remote Send the CLSQR command to the remote RDB server.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**DSS Carrier: RQSDSS****Target System Processing**

The optional *cmdsrid* parameter uniquely identifies the source of the query.

The *monitor* parameter allows the requester to request a set of events or activities to be monitored on the target server. Refer to MONITOR for the list of supported events.

The *pkgnamcsn* parameter specifies the fully qualified package name, the package consistency token, and a section number within the package. The *pkgnamcsn* must match the *pkgnamcsn* parameter of the OPNQR command that opened the query.

The *qryclsrls* parameter specifies whether the target server should release all read locks that are held by the query being closed explicitly. The target server may not necessarily be able to release all read locks held by the query even when requested to do so by the source server since they may be held for other operations or activities.

The *qryinsid* parameter specifies the instance of the query in use. Its value must match the *qryinsid* parameter returned on the OPNQRMR reply message for this instance of the query.

The *rdbnam* parameter specifies the name of the relational database (RDB) that the ACCRDB command accesses. If the *rdbnam* parameter is specified, its value must be the same as the value specified on the ACCRDB command for RDBNAM.

The TYPDEFNAM reply data object specifies the name of the data type to data representation mapping definitions that the target SQLAM uses when sending the SQLCARD reply data object for this command.

The TYPDEFOVR reply data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type to data representation mapping definitions for the SQLCARD reply data object for this command.

Normal completion of this command results in an SQLCARD object being returned.

If a reply message (other than ENDQRYRM) and an SQLCARD object is saved by the previous OPNQRY or CNTQRY command, then the saved reply message and SQLCARD object are returned, and the query is terminated. If the saved reply message was an ENDQRYRM, the saved SQLCARD and the ENDQRYRM are discarded; an appropriate SQLCARD is returned, and the query is terminated.

Table 3-4 (on page 167) is a decision table that summarizes the CLSQRY command's actions.

### Data Conversion

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

### Exceptions

Exception conditions the RDB detects are reported in an SQLCARD object.

If an error condition is detected before the SQLAM initiates the CLSQRY function with the RDB, then the command is rejected with the appropriate reply message, and the query remains in its current state.

If the bind process is active, then the command is rejected with the PKGBPARM, and the query is not changed.

If the query is not open, then the command is rejected with the QRYNOPRM.

If the specified RDB is not currently accessed, then the command is rejected with the RDBNACRM.

### Summary Decision Table

The CLSQRY functions are summarized in the following decision table (Table 3-4 (on page 167)). The decision table only contains information already presented in the previous text. The decision table has been included as a development aid.

In the table, a column specifies a case. Each case contains a set of conditions and a set of actions. An *X* indicates that the condition exists, and a *Y* indicates the resulting actions. For instance, case A has only one condition before the command was received. It is *the query is in the not-opened state*, and the resulting actions are *the QRYNOPRM is returned*, and *the query remains in the not-opened state*.

Table 3-4 CLSQR Summary Decision Table

Conditions	Cases								
	A	B	C	D	E	F	G	H	I
Query is in the not-opened state	X	X	X						
Query is in the suspended state				X	X	X	X	X	X
Package binding process is active		X			X				
ENDQRYRM and SQLCARD were saved by the preceding OPNQRY or CNTQRY command						X			
Reply message (other than ENDQRYRM) and SQLCARD were saved by the preceding OPNQRY or CNTQRY command							X		
An error was detected preventing the CLSQR command from being initiated by the SQLAM			X					X	
RDB had an error preventing the query from closing the database cursor									X
Actions	A	B	C	D	E	F	G	H	I
Query is in the suspended state					Y			Y	
SQLCARD is returned				Y					Y
Saved reply message and SQLCARD are returned							Y		
SQLCARD is returned and discard the saved ENDQRYRM and SQLCARD						Y			
QRYNOPRM is returned	Y								
Query is terminated, placed in the not-opened state	Y	Y	Y	Y		Y	Y		Y
PKGBPARAM is returned		Y			Y				
Reply message is returned; see <code>cmdrpy</code>			Y					Y	

### Source System Reply Processing

Return any reply messages and data objects to the requester.

### Data Conversion

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

### Package Name and Consistency Token

With SQLAM level 7, the PKGNAMCSN is changed to an optional instance variable. If specified, the values are used as the default package name and consistency token for the connection. If not specified on the command, the PKGSN is required to identify the section number. In this case, the default package name, consistency token and specified section number is used for the request. If PKGNAMCSN was never specified on the connection, a PRCCNVRM is returned.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'2005'	
cmdsrcid	INSTANCE_OF OPTIONAL MINLVL	CMDSRCID - Command Source Identifier  7
monitor	INSTANCE_OF OPTIONAL MINLVL	MONITOR - Monitor Events  7
pkgnamcsn	INSTANCE_OF  MINLVL OPTIONAL MTLEXC	PKGNAMCSN - RDB Package Name, Consistency Token, and Section Number  7 Required if PKGSN not specified. PKGSN
pkgsn	INSTANCE_OF MINLVL OPTIONAL MTLEXC	PKGSN - RDB Package Section Number  7 Required if PKGNAMCSN is not specified. PKGNAMCSN
qryclsrls	INSTANCE_OF OPTIONAL MINLVL	QRYCLSRLS - Query Close Lock Release  7
qryinsid	INSTANCE_OF REQUIRED MINLVL	QRYINSID - Query Instance Identifier  7
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>cmddta</b>	<b>NIL</b>	
<b>rpydta</b>	<b>REPLY OBJECTS</b>	
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL MINLVL ENUVAL MINLVL OPTIONAL DFTVAL NOTE	TYPDEFNAM - Data Type Definition Name 'QTDSQL370' 'QTDSQL400' 'QTDSQLX86' 'QTDSQLASC'  4 'QTDSQLVAX'  4  " The default means the value received on ACCRDBRM is used.

X'0035'	INSTANCE_OF OPTIONAL DFTVAL NOTE	TYPDEFOVR - TYPDEF Overrides " The default means the value received on ACCRDBRM is used.
X'1C00'	INSTANCE_OF OPTIONAL MINLVL	MONITORRD - Monitor Reply Data 7
X'1C03'	INSTANCE_OF OPTIONAL MINLVL	MGRVLVLOVR - Manager Level Overrides 7
X'2408'	INSTANCE_OF REQUIRED	SQLCARD - SQL Communications Area Reply Data
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'220D'	INSTANCE_OF NOTE	ABNUOWRM - Abnormal End Unit of Work Condition Returned only if an SQLAM issues the command.
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported
X'2209'	INSTANCE_OF	PKGBPARAM - RDB Package Binding Process Active
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'2202'	INSTANCE_OF	QRYNOPRM - Query Not Open
X'2204'	INSTANCE_OF	RDBNACRM - RDB Not Accessed
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limits Reached
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'125F'	INSTANCE_OF	TRGNSPRM - Target Not Supported
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

**SEE ALSO**

<b>insvar</b>	<i>QRYCLSRLS</i> (on page 682) <i>QRYINSID</i> (on page 688)
<b>inscmd</b>	<i>SQLAM</i> (on page 847)
<b>Semantic</b>	<i>BGNBND</i> (on page 110) <i>CMDSRCID</i> (on page 178) <i>ENDQRYRM</i> (on page 342) <i>FIXROWPRC</i> (on page 417) <i>LMTBLKPRC</i> (on page 475) <i>MONITOR</i> (on page 521) <i>OPNQRY</i> (on page 555) <i>OPNQRYRM</i> (on page 566) <i>QRYCLSIMP</i> (on page 680) <i>QRYCLSRLS</i> (on page 682) <i>QRYINSID</i> (on page 688)

*QRYNOPRM* (on page 690)  
*SQL* (on page 835)  
*SQLAM* (on page 847)



**NAME**

CMDATHRM — Not Authorized to Command

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'121C'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

Not Authorized to Command (CMDATHRM) Reply Message indicates that the user is not authorized to perform the requested command.

**DSS Carrier:** RPYDSS

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'121C'	
<b>rdbnam</b>	INSTANCE_OF MINLVL OPTIONAL	RDBNAM - Relational Database Name 3
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

- cmdrpy** ACCRDB (on page 42)
- ACCSEC (on page 52)
- BGNBND (on page 110)
- BNDSQLSTT (on page 136)
- CLSQRV (on page 165)
- CNTQRY (on page 222)
- DRPPKG (on page 293)
- DSCRDBTBL (on page 300)
- DSCSQLSTT (on page 304)
- ENDBND (on page 336)
- EXCSAT (on page 363)
- EXCSQLIMM (on page 371)
- EXCSQLSET (on page 377)
- EXCSQLSTT (on page 381)
- INTRDBRQS (on page 445)
- OPNQRY (on page 555)
- PRPSQLSTT (on page 636)
- RDBCMM (on page 728)

**rpydta**

*RDBRLLBCK* (on page 745)  
*REBIND* (on page 753)  
*SECCHK* (on page 800)  
*COMMAND* (on page 240)

**NAME**

CMDCHKRM — Command Check

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1254'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

Command Check (CMDCHKRM) Reply Message indicates that the requested command encountered an unarchitected and implementation-specific condition for which there is no architected message. If the severity code value is ERROR or greater, the command has failed. This message can be accompanied by other messages that help to identify the specific condition.

The CMDCHKRM should not be used as a general catch-all in place of product-defined messages when using product extensions to DDM.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'1254'	
<b>rdbnam</b>	INSTANCE_OF MINLVL OPTIONAL NOTE	RDBNAM - Relational Database Name 3  Commands that operate on files do not return this parameter.
<b>reccnt</b>	INSTANCE_OF MINVAL OPTIONAL NOTE  NOTE MINLVL	RECCNT - Record Count 0  Required for requests to insert multiple records in a file. Commands that operate on RDBs do not return this parameter. 3
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL ENUVAL ENUVAL ENUVAL	SVRCOD - Severity Code  0 - INFO - Information Only Severity Code 4 - WARNING - Warning Severity Code 8 - ERROR - Error Severity Code 16 - SEVERE - Severe Error Severity Code

	ENUVAL	32 - ACCDMG - Access Damage Severity Code
	ENUVAL	64 - PRMDMG - Permanent Damage Severity Code
	ENUVAL	128 - SESDMG - Session Damage Severity Code
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

<b>cmdrpy</b>	<p><i>ACCRDB</i> (on page 42)</p> <p><i>ACCSEC</i> (on page 52)</p> <p><i>BGNATMCHN</i> (on page 107)</p> <p><i>BGNBND</i> (on page 110)</p> <p><i>BNDSQLSTT</i> (on page 136)</p> <p><i>CLSQRV</i> (on page 165)</p> <p><i>CNTQRY</i> (on page 222)</p> <p><i>DRPPKG</i> (on page 293)</p> <p><i>DSCRDBTBL</i> (on page 300)</p> <p><i>DSCSQLSTT</i> (on page 304)</p> <p><i>ENDATMCHN</i> (on page 333)</p> <p><i>ENDBND</i> (on page 336)</p> <p><i>EXCSAT</i> (on page 363)</p> <p><i>EXCSQLIMM</i> (on page 371)</p> <p><i>EXCSQLSET</i> (on page 377)</p> <p><i>EXCSQLSTT</i> (on page 381)</p> <p><i>INTRDBRQS</i> (on page 445)</p> <p><i>OPNQRY</i> (on page 555)</p> <p><i>PRPSQLSTT</i> (on page 636)</p> <p><i>RDBCMM</i> (on page 728)</p> <p><i>RDBRLLBCK</i> (on page 745)</p> <p><i>REBIND</i> (on page 753)</p> <p><i>SECCHK</i> (on page 800)</p> <p><i>SYNCCTL</i> (on page 915)</p> <p><i>SYNCRSY</i> (on page 982)</p>
<b>rpydta</b>	<i>COMMAND</i> (on page 240)
<b>Semantic</b>	<p><i>DSS</i> (on page 308)</p> <p><i>EXCSQLSTT</i> (on page 381)</p> <p><i>SYNCPTOV</i> (on page 944)</p>

**NAME**

CMDCMPRM — Command Processing Completed

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'124B'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

Command Processing Completed (CMDCMPRM) Reply Message indicates that the command processing was successfully completed.

The CMDCMPRM is returned only when a command returns no other reply message or reply object.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'124B'	
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code  0 - INFO - Information Only Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**cmdrpy** *INTRDBRQS* (on page 445)

**Semantic** *INTRDBRQS* (on page 445)  
*TCPSRCCD* (on page 1007)

NAME

CMDNSPRM — Command Not Supported

DESCRIPTION (Semantic)

**Dictionary** QDDBASD

**Codepoint** X'1250'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

Command Not Supported (CMDNSPRM) Reply Message indicates that the specified command is not recognized or not supported for the specified target object.

This reply message can be returned only in accordance with the architected rules for DDM subsetting. See *SUBSETS* (on page 902).

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'1250'	
codpnt	INSTANCE_OF REQUIRED NOTE	CODPNT - Codepoint  Specifies the codepoint of the command not supported.
rdbnam	INSTANCE_OF MINLVL OPTIONAL	RDBNAM - Relational Database Name 3
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
svrcod	INSTANCE_OF REQUIRED ENUVAL MINLVL NOTE	SVRCOD - Severity Code  4 - WARNING - Warning Severity Code 2 Can be used if wild-card characters were specified.
	ENUVAL	8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

## SEE ALSO

<b>cmdrpy</b>	<i>ACCRDB</i> (on page 42) <i>ACCSEC</i> (on page 52) <i>BGNATMCHN</i> (on page 107) <i>BGNBND</i> (on page 110) <i>BNDSQLSTT</i> (on page 136) <i>CLSQRY</i> (on page 165) <i>CNTQRY</i> (on page 222) <i>DRPPKG</i> (on page 293) <i>DSCRDBTBL</i> (on page 300) <i>DSCSQLSTT</i> (on page 304) <i>ENDATMCHN</i> (on page 333) <i>ENDBND</i> (on page 336) <i>EXCSAT</i> (on page 363) <i>EXCSQLIMM</i> (on page 371) <i>EXCSQLSET</i> (on page 377) <i>EXCSQLSTT</i> (on page 381) <i>INTRDBRQS</i> (on page 445) <i>OPNQRY</i> (on page 555) <i>PRPSQLSTT</i> (on page 636) <i>RDBCMM</i> (on page 728) <i>RDBRLLBCK</i> (on page 745) <i>REBIND</i> (on page 753) <i>SECCHK</i> (on page 800)
<b>rpydta</b>	<i>COMMAND</i> (on page 240)
<b>Semantic</b>	<i>ACCSEC</i> (on page 52) <i>OPTIONAL</i> (on page 568) <i>SUBSETS</i> (on page 902)

## NAME

CMDSRCID — Command Source Identifier

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'2107'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

Command Source Identifier (CMDSRCID) is generated by the source server to uniquely identify the application source for of any of the following DDM commands: CLSQRY, CNTQRY, DSCSQLSTT, EXCSQLIMM, EXCSQLSTT, OPNQRY, PRPSQLSTT. Each application source may map to a distinct application, or it may be one of many originating from a single application. The source server is responsible for ensuring the CMDSRCID value generated is unique amongst applications sources that are executing DDM commands all multiplexed over a single database connection to the same target server. However, the contents of the CMDSRCID value are implementation-defined and are unarchitected by DDM. Once the CMDSRCID value has been assigned by the source server for an application source, the same CMDSRCID value must be used explicitly or otherwise for all relevant commands stemming from that application source.

The target server uses the CMDSRCID value supplied on a DDM command to distinguish its execution from another execution of the same command from another application source in order to avoid collisions. If the target server receives an incorrect CMDSRCID value for a command which is specified explicitly or otherwise, the command will be executed in the wrong context, in which case an error may or may not result.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	12	
class	X'2107'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	64
	DFTVAL	0
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

## SEE ALSO

**insvar** *CLSQRY* (on page 165)  
*CNTQRY* (on page 222)  
*DSCSQLSTT* (on page 304)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSTT* (on page 381)  
*OPNQRY* (on page 555)  
*PRPSQLSTT* (on page 636)



**NAME**

CMDTRG — Command Target

**DESCRIPTION (Semantic)****Dictionary** QDDPRMD**Codepoint** X'0041'**Length** \***Class** CLASS**Sprcls** OBJECT - Self-identifying Data

Command Target (CMDTRG) Self-identifying Data is a parameter that identifies the object receiving the command. Routing of the command to a command processing program is then a function of both the command's codepoint identifier and the class of the target object.

Every command must have one and only one parameter with the CMDTRG attribute.

See *AGNCMDPR* (on page 64) for a description of command routing.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
<b>length</b>	<b>*</b>
<b>class</b>	<b>X'0041'</b>
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO**

**insvar** *ACCRDB* (on page 42)  
*ACCSEC* (on page 52)  
*BGNBND* (on page 110)  
*BNDSQLSTT* (on page 136)  
*CLSQR* (on page 165)  
*CNTQR* (on page 222)  
*DRPPKG* (on page 293)  
*DSCRDBTBL* (on page 300)  
*DSCSQLSTT* (on page 304)  
*ENDBND* (on page 336)  
*EXCSAT* (on page 363)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSET* (on page 377)  
*EXCSQLSTT* (on page 381)  
*INTRDBRQS* (on page 445)  
*OPNQR* (on page 555)  
*PRPSQLSTT* (on page 636)  
*RDBCMM* (on page 728)  
*RDBRLLBCK* (on page 745)  
*REBIND* (on page 753)  
*SECCHK* (on page 800)  
*SYNCCTL* (on page 915)  
*SYNCRSY* (on page 982)

**Semantic**

*COMMAND* (on page 240)  
*SUBSETS* (on page 902)

**NAME**

CMDVLTRM — Command Violation

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'221D'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

Command Violation (CMDVLTRM) Reply Message indicates that a DDM command violating the processing capabilities of the conversation has been received.

For example, a commitment command (RDBCMM or RDBRLLBCK) has been received on a protected conversation. The RDBCMM and RDBRLLBCK commands are not allowed on protected conversations.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'221D'	
<b>rdbnam</b>	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**cmdrpy** *RDBCMM* (on page 728)  
*RDBRLLBCK* (on page 745)

**Semantic** *RDBCMM* (on page 728)  
*RDBRLLBCK* (on page 745)

**NAME**

CMMRQSRM — Commitment Request

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2225'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

Commitment Request (CMMRQSRM) Reply Message indicates that a dynamic commit or rollback was attempted at the target relational database.

The *cmmtyp* parameter specifies the type of the request (commit or rollback).

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'2225'	
<b>cmmtyp</b>	INSTANCE_OF REQUIRED	CMMTYP - Commitment Request Type
<b>rdbnam</b>	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**cmdrpy** *EXCSQLIMM* (on page 371)  
*EXCSQLSTT* (on page 381)

**Semantic** *EXCSQLIMM* (on page 371)  
*EXCSQLSTT* (on page 381)

**NAME**

CMMTYP — Commitment Request Type

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2143'

**Length** \*

**Class** CLASS

**Sprcls** SCALAR - Scalar Object

Commitment Request Type (CMMTYP) Scalar Object specifies the type of commitment request. A value of one indicates a commit was requested while two indicates a rollback was requested.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	5	
class	X'2143'	
value	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	LENGTH	2
	ENUVAL	X'01'
	NOTE	This value means commit was requested.
	ENUVAL	X'02'
	NOTE	This value means rollback was requested.
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** CMMRQSRM (on page 182)

**NAME**

CMNAPPC — LU 6.2 Conversational Communications Manager

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1444'

**Length** \*

**Class** CLASS

**Sprcls** CMNMGR - Communications Manager

LU 6.2 Conversational Communications Manager (CMNAPPC) describes the communications manager that supports conversational protocols by using Systems Network Architecture Logical Unit 6.2 (SNA LU 6.2) local communications facilities. More information on SNA LU 6.2 is in the *SNA Transaction Programmer's Reference Manual for LU Type 6.2* (GC30-3084, IBM).

CMNAPPC uses the *base* SNA LU 6.2 protocols. CMNAPPC also requires the support of the following SNA LU 6.2 option sets:<sup>2</sup>

- Session-level LU-LU verification (option set 211)
- User ID verification (option set 212)
- Program-supplied user ID and password (option set 213)
- Accounting (option set 243)

If, however, two systems have communications connectivity that includes additional SNA LU 6.2 option sets, then DDM does not prevent using those functions.

**Requirements**

The CMNAPPC conversational protocols used with SNA LU 6.2 are required to do the following:

1. Invoke the target communications manager (TCM) and the target agent process or processes on the target system.
2. Manage the orderly exchange of information by passing the *right to send* between the source communications manager (SCM) and the TCM. This avoids transmission collisions.
3. Send and receive data in the same order as it is transmitted.
4. Detect and handle communications errors in a timely fashion.

---

2. Support of LU 6.2 option sets was not required prior to DDM Level 3.

### Assumptions

The CMNAPPC communications manager assumes that the SNA LU 6.2 session, over which requests, replies, and data can be exchanged, either exists or can be established by the local SNA LU 6.2 communications facility. The local SNA LU 6.2 communications facilities at the source and target systems must establish the physical link, communications path, and the SNA session.

### The Message Envelope Usage

The APPC communications manager accepts commands, replies, and objects (data) from an agent for transmission. The CMNAPPC packages these items into the proper data stream structures.

1. For each command:
  - The CMNAPPC builds an RQSDSS and places the command in it.
  - A new correlation number is generated for the RQSDSS. This correlation number is returned to the source agent.
  - If this command is not the last item to be transmitted, the CMNAPPC sets the chaining bit to *ON* in the RQSDSS.
  - If this command will have command data objects sent with it, the CMNAPPC sets the *next DSS has same request correlator*' bit to *ON* in the DSS.
2. For each reply:
  - The CMNAPPC builds an RPYDSS, if necessary, and places the reply in it. More than one reply can be placed in the same RPYDSS if all of the replies have the same correlation number. The correlation number of the associated RQSDSS is placed in the RPYDSS. The target agent supplies the correlation number.
  - If this reply is not the last item to be transmitted, the CMNAPPC sets the chaining bit to *ON* in the RPYDSS.
  - If this reply will be followed by additional RPYDSSs or OBJDSSs related to the same command, the CMNAPPC sets the *next DSS has same request correlator* bit to *ON* in the DSS.
3. For each object:
  - The CMNAPPC builds an OBJDSS, if necessary, and places the object in it. More than one object can be placed in the same OBJDSS if all of the objects have the same correlation number. The correlation number of the associated RQSDSS is placed in the OBJDSS. The agent supplies the correlation number.
  - If this object is not the last item to be transmitted, the CMNAPPC sets the chaining bit to *ON* in the OBJDSS.
  - If this object will be followed by additional RPYDSSs or OBJDSSs related to the same command, then the CMNAPPC sets the *next DSS has same request correlator* bit to *ON* in the DSS.

The CMNAPPC also passes received commands, replies, and objects to the agent. The CMNAPPC removes the commands, replies, and objects from the RQSDSS, RPYDSS, and OBJDSS structures before passing them to the agent.

### SNA LU 6.2 Verbs Supported

Table 3-5 summarizes the SNA LU 6.2 verbs the CMNAPPC communications manager uses. Local and remote support have the same meanings documented in the *SNA Transaction Programmer's Reference Manual for LU Type 6.2* (GC30-3084, IBM).

**Table 3-5** Communications Manager (CMNAPPC) SNA LU 6.2 Verbs

Verb	Local Support	Remote Support	Notes
ALLOCATE	YES	YES	(1)
CONFIRM	NO	NO	
CONFIRMED	NO	NO	
DEALLOCATE	YES	YES	
FLUSH	YES	YES	
GET_ATTRIBUTES	YES	N/A	(2)
GET_TP_PROPERTIES	YES	N/A	(2)
PREPARE_TO_RECEIVE	NO	N/A	
RECEIVE_AND_WAIT	YES	N/A	
REQUEST_TO_SEND	NO	NO	(3)
SEND_DATA	YES	YES	
SEND_ERROR	YES	YES	

#### Notes

1. The ALLOCATE verb the CMNAPPC issues always specifies TYPE(BASIC\_CONVERSATION), SYNC\_LEVEL(NONE), and PIP(NO).  
The *security* parameter specifies either NONE, SAME, or PGM. If NONE is specified, some target servers may reject the allocation when user identification is required before access to their resources is allowed. CMNAPPC allows the use of *already verified* APPC security functions, but it does not require that a target server instance support those functions.
2. An implementation may need these verbs to build a communications manager. The GET\_ATTRIBUTES and GET\_TP\_PROPERTIES verbs can obtain the LUW\_IDENTIFIER and other information needed for accounting functions.
3. If a REQUEST\_TO\_SEND message is received, it is ignored. Receipt of a REQUEST\_TO\_SEND message cannot cause a communications failure.

#### Communications Manager Initiation

The CMNAPPC SCM operates like any other transaction program. That is, it uses the local SNA LU 6.2 communications facility to communicate with the TCM on the remote system. For a sample protocol sequence involving communications initiation, see *APPCMNI* (on page 72).

The SNA LU 6.2 communications facility is responsible for initiating the SNA 6.2 session. Detailed information about SNA LU 6.2 session initiation is in the SNA LU 6.2 manuals.

SNA LU 6.2 sessions can be established as the result of an ALLOCATE request for an SNA LU 6.2 conversation by the SCM. The ALLOCATE request for an SNA LU 6.2 conversation and session is made in response to an application program requesting access to remote data management services. If a session already exists with the remote system and is available, a conversation and that available session are allocated to the SCM. Otherwise, the SNA LU 6.2 communications facility attempts to establish a new session with the remote system.



The hexadecimal-coded transaction program name (TPN) for the CMNAPPC communications manager that uses the CMNAPPC conversational protocols with the SNA LU 6.2 communications facility can be:

1. A TPN that begins with 07F6 (TPN's registered for relational database access)
2. Any non-SNA-registered name assigns the target system to invoke the CMNAPPC communications manager
3. The registered DDM TPN 07F0F0F1

Once the local SNA LU 6.2 communications facility receives the TPN, the SNA conversation attaches to an instance of the CMNAPPC TCM.

If a conversation and session are successfully allocated, an LU 6.2 conversation with the TCM is established and the target's data management services are made available to the source's application program. The two communications managers communicate by *sending* and *receiving* data structures called data stream structures (DSSs). More information about DSSs can be found in *OBJDSS* (on page 536), *RPYDSS* (on page 766), and *RQSDSS* (on page 774).

If a conversation and a session cannot be established, the ALLOCATE request fails, and the application program is notified that access to the target data management services is not available.

At any time following the successful completion of the SNA LU 6.2 ALLOCATE verb, the SCM can issue a GET\_ATTRIBUTES verb or a GET\_TP\_PROPERTIES verb to obtain information necessary to perform accounting or serviceability functions. At any time following the successful completion of the SNA LU 6.2 attach, the TCM can issue a GET\_ATTRIBUTES verb or a GET\_TP\_PROPERTIES verb to obtain information necessary to perform accounting or serviceability functions. CMNAPPC does not require the SCM or TCM to issue the GET\_ATTRIBUTES verb or GET\_TP\_PROPERTIES verb if accounting and serviceability functions can be fulfilled in other ways.

### CMNAPPC Conversational Protocol

The CMNAPPC conversational protocol requires that the SCM and TCM take turns sending data structures during an SNA conversation. The communications manager whose turn it is to send data structures has the *right to send* for the conversation. The sending communications manager gives up the right to send by passing the *right to send indicator* for the conversation to the receiving communications manager. If this orderly protocol is not followed, information can be lost or damaged, and the two systems will not be able to communicate successfully.

To ensure the orderly exchange of requests and replies, the following CMNAPPC conversational protocol MUST be obeyed:

1. The SCM must not send an RPYDSS to the TCM.
2. The SCM can send only one RQSDSS or RQSDSS chain before waiting for an RPYDSS or OBJDSS from the TCM.
3. The TCM must not send an RQSDSS to the SCM.
4. The TCM can send only one single RPYDSS, RPYDSS chain, single OBJDSS, or OBJDSS chain before waiting for the next RQSDSS from the SCM.
5. The first RQSDSS the SCM sends must contain an EXCSAT command.

### Basic Protocol Flow

When the user application initially requests remote data management services either through its local data manager (LDM), or directly to DDM, connectivity must first be established between the SCM and the TCM. A unique instance of a source agent, and possibly a communications manager, is created on the source system. The SCM (ALLOCATE verb) of the local source communications facility requests an SNA LU 6.2 conversation with the TCM. The SNA LU 6.2 conversation is maintained throughout the life of the source agent. The target agent's TCM does not terminate the conversation.

Normally, more than one SCM/TCM conversation for each user application is not necessary. For example, if a requester on the source system wants to access three files on the target server, then only one SNA LU 6.2 conversation is required between the SCM and TCM. However, a source system implementation is not required to use only one conversation for this situation.

The SCM that acquires an SNA LU 6.2 conversation, has the *right to send* for the conversation. In other words, the source agent is the first party to talk in the conversation through the SCM.

The source agent can then request the SCM to send a chain of RQSDSSs and OBJDSSs through the conversation to the target agent and to pass the *right to send indicator* to the TCM. The SCM then waits for the target to send a chain of RPYDSSs and OBJDSSs and to return the right to send indicator.

When the target agent receives a command, it locates the *target* of each command and passes the command to the command target for execution. Replies returned to the target agent from the command target are sent to the TCM where they are queued. When the TCM has the right to send, it sends all replies to the SCM and passes the right to send indicator for the conversation to the SCM.

For a sample protocol sequence that involves sending a command that has an associated command data object and receiving a single reply message or reply data object in return, see *APPSRCCD* (on page 79). For a sample protocol sequence that involves sending a single command and receiving several reply data objects in return, see *APPSRCCR* (on page 86).

The basic pattern of the source agent sending commands and the target agent sending replies or data is repeated until the application program no longer needs remote data management services from the target system. Now, the source agent requests the SCM to terminate the conversation with the TCM. The SCM uses the local communications facility to terminate the conversation with the TCM (DEALLOCATE verb). The DEALLOCATE verb terminates the SCM/TCM conversation.

### Normal Communications Termination

Under normal circumstances, only the SCM can terminate the conversation between the SCM and the TCM. The SCM should only terminate the conversation when the source system has completed all of its work with the target system. For a sample protocol sequence that involves normal communications termination, see *APPCMNT* (on page 76).

When the SCM deallocates the conversation, the TCM notifies the target agent so that the agent can notify the other server managers to perform any required cleanup. For servers that support files, this cleanup includes:

- Releasing all record and stream locks that are being held
- Closing files that are still open
- Releasing all file locks that are still held

- Performing any additional cleanup, such as freeing up DCLFIL associations or cleaning up internal tables or control blocks

For servers that support RDBs, this cleanup includes:

- Performing a rollback if an RDB is currently accessed and a unit of work is in progress
- Terminating any SQLAM that is currently bound to the agent
- Performing any additional cleanup, such as cleaning up internal tables or control blocks

The SCM terminates without waiting to see if the TCM terminates normally. Therefore, the TCM cannot send an error message if all of the work it is performing has not been completed.

The SNA LU 6.2 session may or may not be terminated when the conversation is terminated. This is up to the SNA LU 6.2 communications facility and cannot be directly controlled by the CMNAPPC communications manager.

### Source Manager-Detected Error

When the SCM processes the DSSs received from the TCM, an error may be detected by the SCM, source agent, or other source manager that prevents the DSS contents from being processed. The error could be that the TCM sent structures that the SCM did not expect.

The basic protocol for handling this type of situation is for the SCM to issue a SEND\_ERROR verb to the SNA LU 6.2 communications facility, and if the SCM detected the error, to notify the source agent that an error was detected. When the SEND\_ERROR verb completes, the SCM is in the send state, and the TCM is in the receive state. For a sample protocol sequence that involves a source manager-detected error, see *APPSRCER* (on page 91).

When the SCM issues a SEND\_ERROR verb, the SCM must conform to one of the following cases:

1. The SCM is in the receive state.

In this case:

- The SCM issues the SEND\_ERROR verb.
- The SCM discards any queued input.

2. The SCM is in the send state.

In this case:

- If the SCM has any queued data ready for output, it may send that data by issuing SEND\_DATA verbs, or it may purge that data. The choice to send or purge depends on the nature of the error that is detected.
- The SCM issues the SEND\_ERROR verb.

These cases mean that the TCM can receive:

- CASE 1: A SEND\_ERROR indication followed by an RQSDSS or a DEALLOCATE.

In this case, the TCM discards any queued output. The TCM then processes the next item received (RQSDSS or DEALLOCATE) normally.

- CASE 2: Various DSSs followed by a SEND\_ERROR indication followed by an RQSDSS or a DEALLOCATE.

In this case, the TCM processes the next item received (RQSDSS or DEALLOCATE) normally.

In either case, the TCM lets the error indication stand. The TCM can log the error or notify someone on the local system, but it is not required.

The source agent is responsible for recovery after a source manager-detected error. The source agent can attempt the command sequence again, return notification to the application program of the error and let it perform recovery, or have the SCM terminate the conversation with the TCM based on the assumption that the TCM is inoperative. CMNAPPC does not have an architected recovery protocol.

### Target Manager-Detected Error

When the TCM processes the data structures received from the SCM, an error might be detected that prevents the TCM or target agent from processing the data structure. The error might be that the SCM sent structures that the TCM did not expect.

The basic protocol for handling this type of situation is for the TCM to issue a `SEND_ERROR` verb to the SNA LU 6.2 communications facility, and if the TCM detected the error, to notify the target agent that an error was detected. The `SEND_ERROR` verb causes the TCM and the SNA LU 6.2 communications facility to discard all non-received data. The SCM is notified that the TCM detected an error. When the `SEND_ERROR` verb completes, the TCM is in the send state and the SCM is in the receive state. The TCM then issues a `SEND_DATA` verb that contains the appropriate RPYDSS (command reply) to notify the SCM and source agent explicitly of the error that the TCM encountered. For a sample protocol sequence that involves a target manager-detected error, see *APPTRGER* (on page 95).

When the TCM issues a `SEND_ERROR` verb, it must conform to one of the following three cases:

1. The TCM has queued output and has sent some of the output to the SCM by issuing a `SEND_DATA` verb.

In this case:

- The TCM finishes sending all queued output to the SCM.
  - The TCM issues the `SEND_ERROR` verb.
  - The TCM sends a reply message indicating the cause of the error. The request correlator of the RPYDSS must be the same as the current RQSDSS that the TCM is processing, or it must be the special request correlator value.
2. The TCM has queued output but has not sent any of the output to the SCM.
    - The TCM issues the `SEND_ERROR` verb.
    - The TCM sends all queued DSS output to the SCM, but the DSSs must be complete and obey all syntax and protocol rules of CMNAPPC.
    - The TCM sends a reply message indicating the cause of the error. The request correlator value of the RPYDSS must be either:
      - The same value as that of the last DSS sent to the SCM
      - The same value as that of the current RQSDSS that the TCM is processing
      - The same value as the special request correlator value
  3. The TCM has no queued output.

In this case:

- The TCM issues the `SEND_ERROR` verb.

- The TCM sends a reply message indicating the cause of the error. The request correlator of the RPYDSS must be the same as the current RQSDSS that the TCM is processing, or it must be the special request correlator value.

These cases mean that the SCM can receive:

- CASE 1: Various DSSs preceding a SEND\_ERROR indication which precedes an RPYDSS.
- CASE 2: A SEND\_ERROR indication preceding various DSSs which precede an RPYDSS.
- CASE 3: A SEND\_ERROR indication preceding an RPYDSS.

The source agent is responsible for recovery after such an error. The source agent can attempt the command sequence again, return notification to the application program of the error and let it perform recovery, or request that the SCM terminate the conversation with the TCM based on the assumption that the SCM is inoperative. CMNAPPC does not have an architected recovery protocol.

The TCM can log the error or notify someone on the local system, but it is not required.

### Communications Failures

A communications failure can be caused by an SNA LU 6.2 session protocol error, an SNA LU 6.2 session outage, a communications line failure, a modem failure, a remote system failure, or by failure of many other types. The result is that the SCM and TCM cannot communicate. Do not confuse communications failures with DDM-detected errors that result in a reply message. See the SNA manuals for detailed information about SNA communications failures. For a sample protocol sequence that involves a communications failure, see *APPCMNFL* (on page 68).

When a communications failure occurs, the TCM:

1. Notifies the target agent that a communications failure has occurred.

Upon notification of the failure, the target agent notifies the other server managers to perform required cleanup functions.

For servers that support files, cleanup functions include:

- Completing, if possible, any DDM command processing
- Releasing all record and stream locks being held
- Closing any files that are open
- Releasing all file locks being held
- Performing additional required cleanup such as freeing up DCLFIL associations

For servers that support relational databases, cleanup functions include:

- Performing a rollback on currently accessed RDBs
- Terminating target SQLAM manager instances
- Performing required additional cleanup

2. Deallocates the SNA conversation.
3. Performs additional required cleanup so the SCM can attempt some form of recovery.

When a communications failure occurs, the SCM:

1. Notifies the source agent that a communications failure has occurred.

Upon notification of the failure, the source agent notifies the other server managers to perform cleanup functions.

These cleanup functions include:

- Performing required additional cleanup
- Notifying the requester that a communications failure has occurred

2. Deallocates the SNA LU 6.2 conversation that exists between the SCM and the TCM.

Reestablishment of the communications connectivity between the source and target systems is outside the scope of the DDM architecture. The SCM can reestablish communications with the TCM by allocating a new SNA LU 6.2 communications conversation (see *APPCMNI* (on page 72)).

**Protocol Flow Sequences**

The following terms illustrate the SNA LU 6.2 protocols and CMNAPPC conversational protocols implementing this communications manager.

APPCMNI	LU 6.2 Communications Initiation (see <i>APPCMNI</i> (on page 72))
APPCMNT	LU 6.2 Communications Termination (see <i>APPCMNT</i> (on page 76))
APPSRCCD	LU 6.2 Source Command with Data (see <i>APPSRCCD</i> (on page 79))
APPSRCCR	LU 6.2 Source Command Returning Data (see <i>APPSRCCR</i> (on page 86))
APPSRCER	LU 6.2 Source Detected Error (see <i>APPSRCER</i> (on page 91))
APPTRGER	LU 6.2 Target Detected Error (see <i>APPTRGER</i> (on page 95))
APPCMNFL	LU 6.2 Communications Failure (see <i>APPCMNFL</i> (on page 68))

**Transaction Program Names**

- 07F0F0F1<sup>3</sup>
- All transaction program names beginning with 07F6
- A non-SNA-registered transaction program name the target system assigns to invoke the CMNAPPC communications manager

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>NIL</b>
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>
<b>mgrlvln</b>	<b>3<sup>5</sup></b>
<b>mgrdepls</b>	<b>NIL</b>

---

3. 07F0F0F1 was the only valid transaction program name (TPN) in DDM prior to DDM Level 3.

<b>vldattls</b>	<b>VALID ATTRIBUTES</b>	
X'0019'	INSTANCE_OF	HELP - Help Text
X'1452'	INSTANCE_OF	MGRNAM - Manager Name
X'0045'	INSTANCE_OF	TITLE - Title

**SEE ALSO**

<b>insvar</b>	<i>MGRLVL</i> (on page 506)
<b>mgrdepls</b>	<i>RSYNMGR</i> (on page 787) <i>SYNCPTMGR</i> (on page 939) <i>XAMGR</i> (on page 1063)
<b>Semantic</b>	<i>ACCRDB</i> (on page 42) <i>AGENT</i> (on page 61) <i>APPCMNI</i> (on page 72) <i>CMNMGR</i> (on page 196) <i>CMNOVR</i> (on page 201) <i>CMNSYNCPT</i> (on page 202) <i>INHERITANCE</i> (on page 437) <i>RPYDSS</i> (on page 766) <i>RQSDSS</i> (on page 774) <i>RSYNMGR</i> (on page 787) <i>SQLAM</i> (on page 847) <i>SUBSETS</i> (on page 902) <i>SYNCPTMGR</i> (on page 939) <i>SYNCPTOV</i> (on page 944)

---

5. DDM Level 2 and DDM Level 1 CMNAPPC were functionally identical. DDM Level 3 added new functions.

**NAME**

CMNLYR — Communications Layers

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

Communications Layers (CMNLYR) describes the layers of the DDM architecture from the communications point of view. Figure 3-16 (on page 195) illustrates these layers. The functions of each layer are described in the figure, but it is important to note that each layer has a component in both the source server and the target server. The diagram shows that the application program interacts directly with the file (for file services) as application data flows between them.

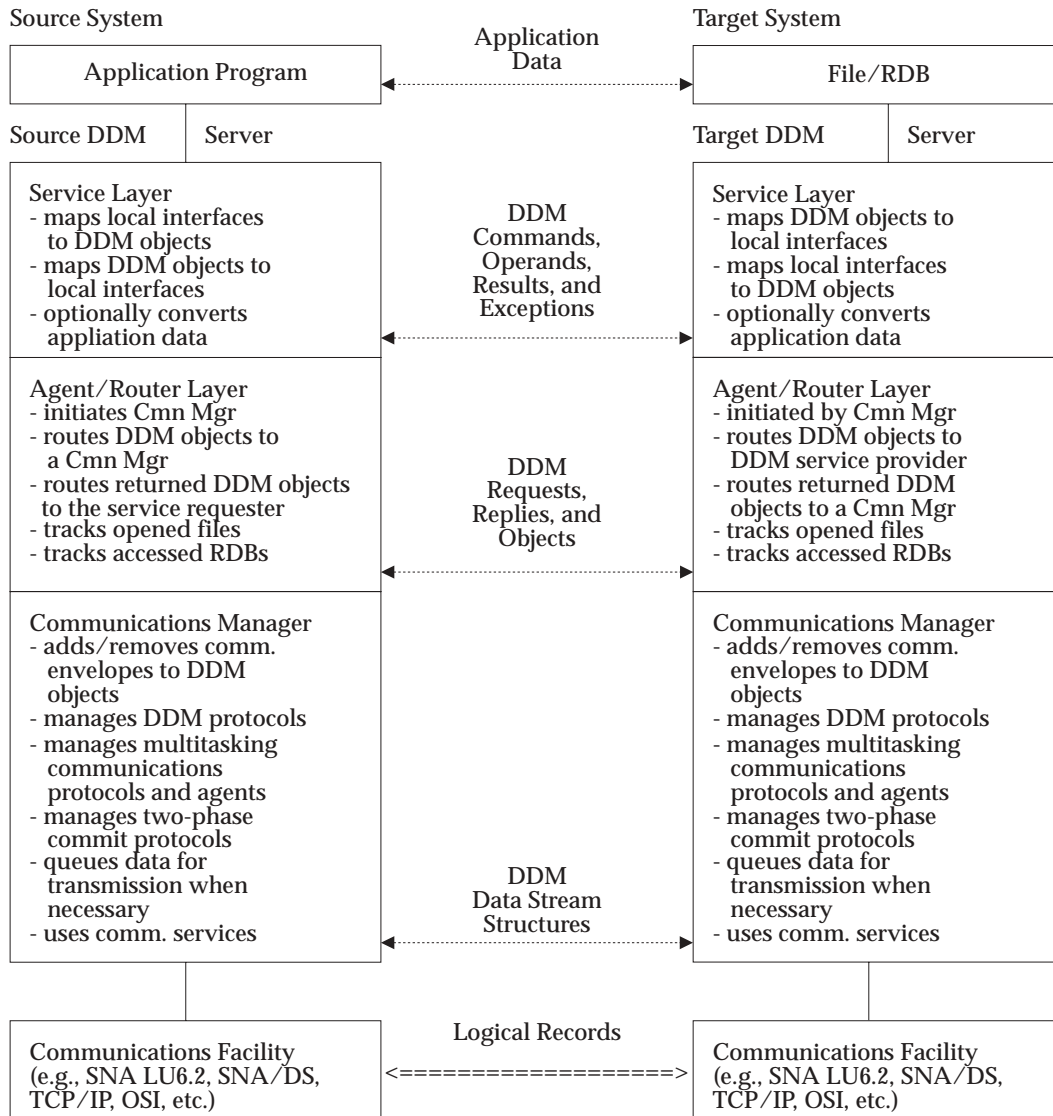
While all communications must, of course, flow through the communications facility in use (for example, SNA LU 6.2), logical communications are designed to flow between the source and target components of each layer. This allows the service and agent layers of DDM to function independent of the communications facility being used. The conversational use of SNA LU 6.2 has been documented in DDM Levels 1-4, but other communications facilities have been used (SNA LU 7) and could be used (for example, TCP/IP or OSI). A DDM communications manager implements the DDM conversational protocol on whatever communications facility is being used. Other DDM protocols could also be implemented, for example, based on the use of full-duplex communications or on the use of asynchronous SNA/FS communications.

Because of this layering, the source service requester appears to interact directly with the target service providers, the source agent appears to interact directly with the target agent, and the source communications manager appears to interact directly with the target communications manager. The interactions at each layer are through messages designed specifically for the layer, with lower-layer messages enveloping the messages of the next higher layer.

**Data Conversion**

Only application data is converted. There is no need to convert the envelopes of the communications manager or agent layers of DDM. A single set of data representations is used for all agent and communications manager data.





**Figure 3-16** DDM as a Layered Communications Architecture

**SEE ALSO**

**Semantic** *CMNOVR* (on page 201)  
*CONCEPTS* (on page 243)

NAME

CMNMGR — Communications Manager

DESCRIPTION (Semantic)

Dictionary QDDBASD

Codepoint X'1408'

Length \*

Class CLASS

Sprcls MANAGER - Resource Manager

Communications Manager (CMNMGR) is one of the basic operative parts of the DDM architecture. Its position in the server data paths is shown in Figure 3-17.

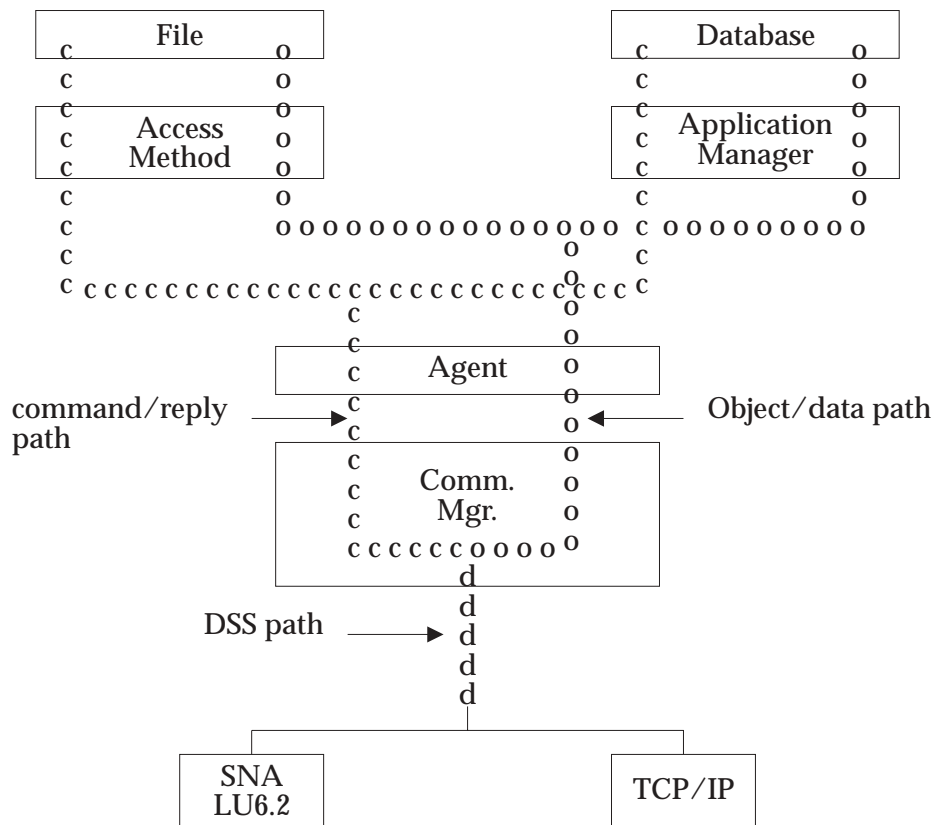


Figure 3-17 Server Data Paths for Access Method Services

The primary functions of a communications manager are:

1. Interfacing with the local communications facility (LCF) to receive and send DDM communications messages, requests, replies, and data (records).

This interface is not formally defined in DDM because it can vary according to the LCF being used. See *CMNAPPC* (on page 184) for using SNA LU 6.2 for the DDM conversational protocol. See *CMNSYNCPT* (on page 202) to find out how to use SNA LU 6.2 and sync point for the DDM conversational protocol. See *CMNTCPIP* (on page 214) to find out how to use TCP/IP for the DDM conversational protocol.

The communications manager assumes that a logical connection exists over which communications messages, requests, replies, and data can be exchanged. The LCF at the source and target systems must establish the physical link, communications path, and logical connection (SNA session or TCP/IP connection).

2. Routing the requests and replies received from the local communications facility to the proper agent.

Each instance of a communications manager can interface to one or more agents. Implementations are not required to support multiple agents per instance of a communications manager. This allows a single communications manager to interface with the local communications facility for several requesters.

3. Interfacing with DDM agents and other objects.

The communications manager accepts commands, replies, and objects (data) from an agent for transmission. The communications manager packages these items into the proper data stream structures.

1. For each command:
  - The communications manager builds an RQSDSS and places the command in it.
  - A new correlation number is generated for the RQSDSS. This correlation number is returned to the source agent.
  - If this command is not the last item to be transmitted, the communications manager sets the chaining bit to *ON* in the RQSDSS.
  - If this command will have command data objects sent with it, the communications manager sets the *next DSS has same request correlator* bit to *ON* in the DSS.
2. For each reply:
  - The communications manager builds a RPYDSS, if necessary, and places the reply in it. More than one reply can be placed in the same RPYDSS if all of the replies have the same correlation number. The correlation number of the associated RQSDSS is placed in the RPYDSS. The target agent supplies the correlation number.
  - If this reply is not the last item to be transmitted, the communications manager sets the chaining bit to *ON* in the RPYDSS.
  - If this reply will be followed by additional RPYDSSs or OBJDSSs related to the same command, the communications manager sets the *next DSS has same request correlator* bit to *ON* in the DSS.
3. For each object:
  - The communications manager builds an OBJDSS, if necessary, and places the object in it. More than one object can be placed in the same OBJDSS if all of the objects have the same correlation number. The correlation number of the associated RQSDSS is placed in the OBJDSS. The target agent supplies the correlation number.
  - If this object is not the last item to be transmitted, the communications manager sets the chaining bit to *ON* in the OBJDSS.
  - If this object precedes additional RPYDSSs or OBJDSSs related to the same command, the communications manager sets the *next DSS has same request*

*correlator* bit to *ON* in the DSS.

- If this object is encrypted, the communications manager sets the encryption bit to *ON* in the OBJDSS.
- If this object is to be encrypted, the communications manager accesses the security services to encrypt the object and builds an Encrypted OBJDSS.
- If this object is to be decrypted, the communications manager accesses the security services to decrypt the Encrypted OBJDSS.

4. For each communications message:

- The multitasking communications manager builds a CMNDSS of the appropriate subtype.
- The proper execution priority of the agent is set in the CMNDSS.

The communications manager removes the commands, replies, and objects from the RQSDSS, RPYDSS, and OBJDSS structures before passing them to the agent.

4. The communications manager queues output whenever the local communications facility is not ready to accept data for transmission. For example, in SNA the local communications facility will not accept data for transmission until it is in the send state.

The communications manager queues output until it is able to pass the DSS to the local communications facility for transmission. If the output (send) queue runs out of space before the communications manager can pass the queued output to the communications facility, the communications manager:

1. Discards any DSS that does not completely fit onto the output queue (only complete DDM objects can be transmitted).
2. Takes action to cause all queued input to be discarded (purged).
3. When the communications facility can accept output from the communications manager, the queued output is passed to the communications facility followed by a RPYDSS containing a protocol error reply message indicating a send-buffer-overflow condition occurred.

5. The communications manager preserves the order of all objects it receives. Commands, replies, and objects received from an agent are placed in DSS structures and sent to the communications facility in the same order that the communications manager received them. The communications manager passes commands, replies, and objects to the operative parts of a DDM implementation in the same order they were received from the communications facility.

6. The interface to DDM communications managers includes commands to:

1. Initiate communications
2. Terminate communications
3. Initiate commit or rollback processing
4. Send RQSDSSs
5. Send request OBJDSSs (records)
6. Receive RQSDSSs
7. Receive request OBJDSSs

8. Send RPYDSSs
  9. Send reply OBJDSSs
  10. Receive RPYDSSs
  11. Receive reply OBJDSSs
  12. Retrieve attributes of the communications facility and the current communications connection
  13. Send and receive CMNDSSs<sup>6</sup>
  14. Encrypt request and reply OBJDSSs
  15. Decrypt request and reply OBJDSSs
7. The target communications manager queues OBJDSSs associated with an RQSDSS separately. Data (contained in an OBJDSS) that is associated with an RQSDSS is not passed directly to the agent, but is retained by the communications manager until the agent requests the data.
8. Detecting normal and abnormal communications termination.

When the communications manager detects that communications have terminated (normally or abnormally), the communications manager:

1. Passes notification to the agent that communications have terminated so that the proper cleanup (releasing locks, closing files, and so on) can be performed.
  2. Discards any requests, replies, or queued data the communications manager or local communications facility holds.
  3. Removes the routing information from the communications manager routing table for the communications address that was terminated.
  4. Deletes the instance of the agent associated with the terminated communications.
9. Enforcing the DDM communications protocol rules.

Each instance of a communications manager supports only one of the DDM communications protocols for one local communications facility (SNA or TCP/IP).

The defined DDM communications protocols are:

CMNAPPC	SNA LU 6.2 Conversational Protocol (see <i>CMNAPPC</i> (on page 184))
CMNSYNCPT	SNA LU 6.2 Sync Point Conversational Protocol (see <i>CMNSYNCPT</i> (on page 202))
CMNTCPIP	TCP/IP Conversational Protocol (see <i>CMNTCPIP</i> (on page 214))

Other DDM communications protocols will be defined as needed.

---

6. The multitasking communications manager is introduced in DDM Level 4.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>NIL</b>	
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>mgrlvl</b>	<b>1</b>	
<b>mgrdepls</b>	<b>NIL</b>	
<b>vldattls</b>	<b>VALID ATTRIBUTES</b>	
X'0019'	INSTANCE_OF	HELP - Help Text
X'1452'	INSTANCE_OF	MGRNAM - Manager Name
X'0045'	INSTANCE_OF	TITLE - Title

**SEE ALSO**

<b>Semantic</b>	<i>CMNAPPC</i> (on page 184)
	<i>CMNOVR</i> (on page 201)
	<i>CMNSYNCPT</i> (on page 202)
	<i>CMNTCPIP</i> (on page 214)
	<i>EXTENSIONS</i> (on page 400)
	<i>INHERITANCE</i> (on page 437)
	<i>RDBOVR</i> (on page 740)

**NAME**

CMNOVR — Communications Overview

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

Communications Overview (CMNOVR) discusses the many different types of communications facilities which transmit and receive DDM data stream structures. DDM ensures data connectivity between different implementations. To do this, a limited number of DDM protocols are defined as part of the DDM architecture. Each DDM protocol is designed for a specific communications environment and defines DDM communications in that environment. Programmers who are implementing products can select the environments and the protocols being supported.

CMNLYR            Communications Layers (see *CMNLYR* (on page 194))

CMNMGR            Communications Manager (see *CMNMGR* (on page 196))

CMNAPPC            LU 6.2 Conversational Communications Manager (see *CMNAPPC* (on page 184))

CMNSYNCPT        SNA LU 6.2 Sync Point Conversational Communications Manager (see *CMNSYNCPT* (on page 202))

CMNTCPIP            TCP/IP Communication Manager (see *CMNTCPIP* (on page 214))

**SEE ALSO**

**Semantic**            *MGROVR* (on page 514)

**NAME**

CMNSYNCPT — SNA LU 6.2 Sync Point Conversational Communications Manager

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'147C'

**Length** \*

**Class** CLASS

**Sprcls** CMNMGR - Communications Manager

SNA LU 6.2 Sync Point Conversational Communications Manager (CMNSYNCPT) provides an SNA LU 6.2 Conversational Communications Manager (CMNAPPC) with sync point support (CMNSYNCPT). For more information on SNA LU 6.2, see the *SNA Transaction Programmer's Reference Manual for LU Type 6.2* (GC30-3084, IBM).

CMNSYNCPT uses the *base* SNA LU 6.2 protocols. CMNSYNCPT also requires the support of the following SNA LU 6.2 option sets:

- Session-level LU-LU verification (option set 211)
- User ID verification (option set 212)
- Program-supplied user ID and password (option set 213)
- Accounting (option set 243)
- Sync point services (option set 108)

If, however, two systems have communications connectivity including additional SNA LU 6.2 option sets, CMNSYNCPT does not preclude use of those functions.

**Requirements**

The DDM conversational protocols for SNA LU 6.2 sync point enable the CMNSYNCPT to:

1. Invoke the target communications manager (TCM) and the target agent process or processes on the target system.
2. Manage the orderly exchange of information by passing the *right to send* between the source communications manager (SCM) and TCM, thus avoiding transmission collisions.
3. Send and receive data in the same order as it is transmitted.
4. Detect and handle communications errors in a timely fashion.
5. Participate in two-phase commit protocols to ensure coordinated resource recovery.

**Assumptions**

The CMNSYNCPT assumes the SNA LU 6.2 session, over which requests, replies, and data can be exchanged, either exists or can be established by the local SNA LU 6.2 communications facility. The LU 6.2 communications facilities at the source and target systems must establish the physical link, communications path, and the SNA session.



### The Message Envelope Usage

The APPC communications manager accepts commands, replies, and objects (data) from an agent for transmission. The CMNAPPC packages these items into the proper data stream structures.

1. For each command:
  - The CMNAPPC builds an RQSDSS and places the command in it.
  - A new correlation number is generated for the RQSDSS. This correlation number is returned to the source agent.
  - If this command is not the last item to be transmitted, the CMNAPPC sets the chaining bit to *ON* in the RQSDSS.
  - If this command will have command data objects sent with it, the CMNAPPC sets the *next DSS has same request correlator* bit to *ON* in the DSS.
2. For each reply:
  - The CMNAPPC builds a RPYDSS, if necessary, and places the reply in it. More than one reply can be placed in the same RPYDSS if all of the replies have the same correlation number. The correlation number of the associated RQSDSS is placed in the RPYDSS. The target agent supplies the correlation number.
  - If this reply is not the last item to be transmitted, the CMNAPPC sets the chaining bit to *ON* in the RPYDSS.
  - If this reply will be followed by additional RPYDSSs or OBJDSSs related to the same command, the CMNAPPC sets the *next DSS has same request correlator* bit to *ON* in the DSS.
3. For each object:
  - The CMNAPPC builds an OBJDSS, if necessary, and places the object in it. More than one object can be placed in the same OBJDSS if all of the objects have the same correlation number. The correlation number of the associated RQSDSS is placed in the OBJDSS. The agent supplies the correlation number.
  - If this object is not the last item to be transmitted, the CMNAPPC sets the chaining bit to *ON* in the OBJDSS.
  - If this object will be followed by additional RPYDSSs or OBJDSSs related to the same command, then the CMNAPPC sets the *next DSS has same request correlator* bit to *ON* in the DSS.

The CMNAPPC also passes received commands, replies, and objects to the agent. The CMNAPPC removes the commands, replies, and objects from the RQSDSS, RPYDSS, and OBJDSS structures before passing them to the agent.

### SNA LU 6.2 Verbs Supported

Table 3-6 (on page 204) summarizes the SNA LU 6.2 verbs that the CMNSYNCPT uses. Local and remote support are defined in the *SNA Transaction Programmer's Reference Manual for LU Type 6.2* (GC30-3084, IBM).

**Table 3-6** Communications Manager (CMNSYNCPT) SNA LU 6.2 Verbs

Verb	Local Support	Remote Support	Notes
ALLOCATE	YES	YES	(1)
BACKOUT	YES	YES	(2)
CONFIRM	NO	NO	
CONFIRMED	NO	NO	
DEALLOCATE	YES	YES	
FLUSH	NO	YES	
GET_ATTRIBUTES	YES	N/A	(3)
GET_TP_PROPERTIES	YES	N/A	(3)
PREPARE_TO_RECEIVE	NO	N/A	
RECEIVE_AND_WAIT	YES	N/A	
REQUEST_TO_SEND	NO	NO	(4)
SEND_DATA	YES	YES	
SEND_ERROR	YES	YES	
SET_SYNCPT_OPTIONS	NO	NO	(5)
SYNCPT	YES	YES	(2)

**Notes**

1. The ALLOCATE verb the CMNSYNCPT issues always specifies TYPE(BASIC\_CONVERSATION), SYNC\_LEVEL(SYNCPT), and PIP(NO).

The *security* parameter specifies either NONE, SAME, or PGM. If NONE is specified, some target servers may reject the allocation when user identification is required before access to their resources is allowed. CMNSYNCPT allows the use of *already verified* APPC security functions, but it does not require that a target server instance support those functions.

2. The communications manager must support the sync point verbs BACKOUT and SYNCPT. See the *SNA Transaction Programmer's Reference Manual for LU Type 6.2*, (GC30-3084, IBM).
3. An implementation may need these verbs to build a communications manager. The GET\_ATTRIBUTES and GET\_TP\_PROPERTIES verbs can be used to obtain the LUW\_IDENTIFIER and other information needed for accounting functions.
4. If a REQUEST\_TO\_SEND message is received, it is ignored. Receipt of a REQUEST\_TO\_SEND message cannot cause a communications failure.
5. The SET\_SYNCPT\_OPTIONS verb supports SNA LU 6.2 option sets that provide synchronization point optimizations. DDM encourages the implementation of this verb but does not rely on or require the implementation of this verb.

If a product chooses to implement the option set that allows a resource to *vote read only* during resource recovery processing, the resource cannot *vote read only* when cursors are held at the resource.

### Communications Manager Initiation

The CMNSYNCPT source communications manager (SCM) operates like any other transaction program. It uses the local SNA LU 6.2 communications facility to communicate with the target communications manager (TCM) on the remote system. For a sample protocol sequence involving communications initiation, see *SYNCMNI* (on page 931).

The SNA LU 6.2 communications facility must initiate the SNA LU 6.2 session. Detailed information about SNA LU 6.2 session initiation is in the SNA LU 6.2 manuals.

SNA LU 6.2 sessions can be established if the SCM issues an ALLOCATE request for an SNA LU 6.2 conversation. The ALLOCATE request for an SNA LU 6.2 conversation and session is made in response to an application program requesting access to remote data management services. If a session already exists with the remote system and is available, a conversation and that available session are allocated to the SCM. Otherwise, the SNA LU 6.2 communications facility attempts to establish a new session with the remote system.

The hexadecimal-coded transaction program name (TPN) for the CMNSYNCPT that uses the CMNSYNCPT conversational protocols with the SNA LU 6.2 sync point communications facility can be one of the following:

- A TPN that begins with 07F6 (a TPN registered for relational database access)
- A non-SNA-registered name the target system assigns to invoke the CMNSYNCPT
- The registered DDM TPN 07F0F0F1

Once the local SNA LU 6.2 communications facility receives the TPN, the SNA conversation attaches to an instance of the CMNSYNCPT TCM.

If a conversation and session are successfully allocated, an SNA LU 6.2 conversation with the TCM is established and the target's data management services are made available to the source's application program. The two communications managers communicate by sending and receiving data stream structures (DSSs). More information about DSSs can be found in *OBJDSS* (on page 536), *RPYDSS* (on page 766), and *RQSDSS* (on page 774).

If a conversation and a session cannot be established, the ALLOCATE verb fails and the application program is notified that access to the target's data management services is not available.

At any time following the successful completion of the SNA LU 6.2 ALLOCATE verb, the SCM can issue a GET\_ATTRIBUTES verb or a GET\_TP\_PROPERTIES verb to obtain information necessary to perform accounting or serviceability functions. At any time following the successful completion of the SNA LU 6.2 attach, the TCM can issue a GET\_ATTRIBUTES verb or a GET\_TP\_PROPERTIES verb to obtain information necessary to perform accounting or serviceability functions. CMNSYNCPT does not require the SCM or TCM to issue the GET\_ATTRIBUTES verb or GET\_TP\_PROPERTIES verb if its accounting and serviceability functions can be fulfilled in other ways.

### CMNSYNCPT Conversational Protocol

The CMNSYNCPT conversational protocol requires that the SCM and TCM take turns sending data structures during an SNA conversation. The communications manager whose turn it is to send data structures has the *right to send* for the conversation. The sending communications manager gives up the *right to send* by passing it to the receiving communications manager. If this orderly protocol is not followed, information can be lost or damaged, and the two systems will not be able to communicate successfully.

To ensure the orderly exchange of requests and replies, the following CMNSYNCPT conversational protocol must be obeyed:

1. The SCM must not send an RPYDSS to the TCM.
2. The SCM can send only one RQSDSS or RQSDSS chain before waiting for an RPYDSS or OBJDSS from the TCM.
3. The TCM must not send an RQSDSS to the SCM.
4. The TCM can send only one single RPYDSS, RPYDSS chain, single OBJDSS, or OBJDSS chain before waiting for the next RQSDSS from the SCM.
5. The first RQSDSS the SCM sends must contain an EXCSAT command.

### Basic Protocol Flow

When the user application initially requests remote data management services either through its local data manager (LDM), or directly to DDM, the SCM must connect with the TCM. A unique instance of a source agent, and possibly a communications manager, is created on the source system. The SCM of the local source communications facility requests an SNA LU 6.2 conversation with the TCM by issuing the ALLOCATE verb. The SNA LU 6.2 communications facilities propagates the source system's unit of work to the target system when the conversation is established. The SNA LU 6.2 conversation is maintained throughout the life of the source agent. Under normal circumstances, the TCM does not terminate the conversation.

Normally, more than one SCM/TCM conversation for each user application is not necessary. For example, if a requester on the source system wants to access three files on the target server, only one SNA LU 6.2 conversation is required between the SCM and TCM. However, a source system implementation is not required to use only one conversation for this situation.

The SCM that acquires an SNA LU 6.2 conversation has the *right to send* for the conversation. In other words, the source agent is the first party to talk in the conversation through the SCM.

The source agent with the *right to send* can request the SCM to send a chain of RQSDSSs and OBJDSSs through the conversation to the target agent and to pass the *right to send* to the TCM. The SCM then waits for the target to send a chain of RPYDSSs and OBJDSSs and to return the *right to send*.

When the target agent receives a command, it locates the target of each command and passes the command to the command target for execution. Replies returned to the target agent from the command target are sent to the TCM, where they are queued. When the TCM has the *right to send*, it sends all replies to the SCM and passes the *right to send* for the conversation to the SCM.

For a sample protocol sequence that involves sending a command that has an associated command data object and receiving a single reply message or reply data object in return, see *APPSRCCD* (on page 79). For a sample protocol sequence that involves sending a single command and receiving several reply data objects in return, see *APPSRCCR* (on page 86).

The basic pattern of the source agent sending commands and data and the target agent sending reply messages and data is repeated until the application program completes what it considers to be a unit of work (UOW). At that time, the application program requests the commitment of the UOW. This request initiates the two-phase commit process. Upon successful completion of the two-phase commit process, the next UOW is automatically started.

If the application program determines that the updates made during the current unit of work are incorrect and should not be made permanent, the application program requests that the unit of work be backed out (by using the BACKOUT verb). This request initiates the backout process. Upon successful completion, updates made as part of the unit of work are removed, and the

next unit of work is automatically started.

The application program can perform a sequence of one or more units of work. When the application program has completed the final unit of work with the target system, the source agent requests the SCM to terminate the conversation with the TCM. The SCM uses the local communications facility to terminate the conversation with the TCM (by using the DEALLOCATE verb). The DEALLOCATE verb, followed by a SYNCPT verb, terminates the SCM/TCM conversation. This SYNCPT verb will commit the last automatically started unit of work which should not have any updates in it.

### Normal Communications Termination

Under normal circumstances, only the SCM causes the conversation between the SCM and the TCM to terminate. The SCM only terminates the conversation when the source system has completed all of its work with the target system. For a sample protocol sequence involving normal communications termination, see *SYNCMNT* (on page 935).

When the SCM deallocates its conversation with the TCM, the TCM notifies the target agent so the agent can notify the other server managers to perform any required cleanup.

The SCM issues a DEALLOCATE TYPE(SYNC\_LEVEL) followed by a SYNCPT verb.

- The TCM receives a TAKE\_SYNCPT\_DEALLOCATE indication and issues a SYNCPT verb to SNA LU 6.2, which in turn notifies the SYNCPTMGR to continue the two-phase commit process.
- After successful completion of the SYNCPT verb, the TCM issues a DEALLOCATE(TYPE(LOCAL)) and then notifies the agent to perform the normal non-recovery cleanup functions.

For servers that support files, cleanup includes:

- Releasing all record and stream locks that are being held
- Closing files that are still open
- Releasing all file locks that are still held
- Performing any additional cleanup such as freeing up DCLFIL associations, or cleaning up internal tables or control blocks

For servers that support relational databases, cleanup includes:

- Terminating any SQLAM that is currently bound to the agent
- Performing any additional cleanup such as cleaning up internal tables or control blocks

The SCM must wait to see if the SYNCPT verb completes successfully to ensure that the conversation is terminated.

The SNA LU 6.2 session may or may not terminate when the conversation is terminated. This is up to the SNA LU 6.2 communications facility which the CMNSYNCPT cannot directly control.

**Source Manager-Detected Error**

When the SCM processes the DSSs received from the TCM, an error may be detected by the SCM, source agent, or other source manager that prevents the DSS contents from being processed. The error might be that the TCM sent structures that the SCM did not expect.

The basic protocol for handling this type of situation is for the SCM to issue a SEND\_ERROR verb to the SNA LU 6.2 communications facility and if the SCM detected the error, to notify the source agent that an error was detected. When the SEND\_ERROR verb completes, the SCM is in the send state and the TCM is in the receive state. For a sample protocol sequence that involves a source manager-detected error, see *APPSRCER* (on page 91).

When the SCM issues a SEND\_ERROR verb, the protocol depends on whether the SCM is in the receive or send state.

1. The SCM is in the receive state.

In this case:

- The SCM issues the SEND\_ERROR verb.
- The SCM discards any queued input.

2. The SCM is in the send state.

In this case:

- If the SCM has any queued data ready for output, it may purge that data, or it may send that data by issuing SEND\_DATA verbs. The choice to purge or send depends on the nature of the error that is detected.
- The SCM issues the SEND\_ERROR verb.

What the TCM can receive also depends on the receive and send state of the SCM.

1. When the SCM is in the receive state:

- The TCM receives SEND\_ERROR indication followed by an RQSDSS or a DEALLOCATE.
- The TCM discards any queued output. The TCM then processes the next item received (RQSDSS or DEALLOCATE) successfully.

2. When the SCM is in the send state:

- The TCM receives various DSSs followed by a SEND\_ERROR indication followed by an RQSDSS or a DEALLOCATE.
- The TCM processes the next item received (RQSDSS or DEALLOCATE) successfully.

The TCM has three options depending on what it receives:

1. Receives RQSDSS—The command within the RQSDSS is part of the current unit of work and processing continues normally.
2. Receives TAKE\_SYNCPT\_DEALLOCATE indication—Processing continues as outlined in the previous section, **Normal Communications Termination** (on page 207).
3. Receives DEALLOCATE\_ABEND indication—The TCM issues a BACKOUT verb, and upon its successful completion, issues a DEALLOCATE(TYPE(LOCAL)) and performs non-recovery cleanup actions as discussed in the previous section, **Normal Communications Termination** (on page 207).

The source agent is responsible for recovery after a source manager-detected error. The source agent can:

1. Attempt the command sequence again.
2. Return notification of the error to the application program and let it perform recovery.
3. Have the SCM terminate the conversation with the TCM based on the assumption that the TCM is inoperative.

CMNSYNCPT does not have an architected recovery protocol.

### Target Manager-Detected Error

When the TCM processes the data structures received from the SCM, an error may be detected that prevents the TCM or target agent from processing the data structure. The error might be that the SCM sent structures that the TCM did not expect.

The basic protocol for handling this type of situation is for the TCM to issue a `SEND_ERROR` verb to the SNA LU 6.2 communications facility and if the TCM detected the error, to notify the target agent that an error was detected. The `SEND_ERROR` verb causes the TCM and the SNA LU 6.2 communications facility to discard all data not received, and the SCM is notified that the TCM detected an error. When the `SEND_ERROR` verb completes, the TCM is in the send state and the SCM is in the receive state. The TCM then issues a `SEND_DATA` verb that contains the appropriate RPYDSS (command reply) to notify the SCM and source agent explicitly of the error that the TCM encountered. For a sample protocol sequence that involves a target manager-detected error, see *APPTRGER* (on page 95).

When the TCM issues a `SEND_ERROR` verb, it must conform to one of the following three cases:

1. If the TCM has queued output and has sent some of the output to the SCM by issuing a `SEND_DATA` verb:
  - The TCM finishes sending all queued output to the SCM.
  - The TCM issues the `SEND_ERROR` verb.
  - The TCM sends a reply message indicating the cause of the error. The request correlator value of the RPYDSS must be the same as the request correlator value of the current RQSDSS that the TCM is processing or the special request correlator value.
2. If the TCM has queued output but has not sent any of the output to the SCM:
  - The TCM issues the `SEND_ERROR` verb.
  - The TCM sends all queued DSS output to the SCM, but the DSSs must be complete and obey all syntax and protocol rules of the CMNSYNCPT.
  - The TCM sends a reply message indicating the cause of the error. The request correlator value of the RPYDSS must be the same value as one of the following:
    - The last DSS sent to the SCM
    - The current RQSDSS that the TCM is processing
    - The special request correlator value
3. If the TCM has no queued output:
  - The TCM issues the `SEND_ERROR` verb.
  - The TCM sends a reply message indicating the cause of the error. The request correlator value of the RPYDSS must be either the same request correlator value as the

current RQSDSS that the TCM is processing or the special request correlator value.

As illustrated by the case numbers from above, the SCM can receive:

- For case 1: various DSSs followed by a SEND\_ERROR indication followed by an RPYDSS
- For case 2: a SEND\_ERROR indication followed by various DSSs followed by an RPYDSS
- For case 3: a SEND\_ERROR indication followed by an RPYDSS

The source agent is responsible for recovery after a target manager-detected error. The source agent can:

1. Attempt the command sequence again.
2. Return notification to the application program of the error and let it perform recovery.
3. Request that the SCM terminate the conversation with the TCM based on the assumption that the SCM is inoperative.

The CMNSYNCPT does not have an architected recovery protocol.

The TCM may log the error or notify someone on the local system, but neither action is required.

### Communications Failures

A communications failure can be caused by an SNA LU 6.2 session protocol error, an SNA LU 6.2 session outage, a communications line failure, a modem failure, a remote system failure, or by failures of many other types, resulting in a communication breakdown between the SCM and the TCM. Do not confuse communications failures with DDM-detected errors that result in a reply message. See the SNA manuals for detailed information about SNA communications failures. For a sample protocol sequence involving a communications failure, see *SYNCMNFL* (on page 928).

If a communications failure occurs and the UOW is in the in doubt state, neither the SCM nor the TCM perform the error recovery outlined in the following sections. Instead, the SNA LU 6.2 communications facility, the SYNCPTMGR, the SQLAM, and the relational database (RDB) work together to accomplish the resynchronization process. See *RESYNOVR* (on page 764) for an overview of the resynchronization process.

The TCM does the following when a communications failure occurs, and the UOW is not in the in doubt state:

1. Issues a BACKOUT verb to the local server.  
For servers that support recoverable resources, this includes performing a backout on currently accessed recoverable resources.
2. Notifies the target agent that a communications failure has occurred.  
Upon notification of the failure, the target agent notifies the other server managers to perform required cleanup functions.  
For servers that support files, cleanup functions include:
  - Completing, if possible, any DDM command processing
  - Releasing all record and stream locks being held
  - Closing any files that are open
  - Releasing all file locks being held



- Performing additional required cleanup, such as freeing up DCLFIL associations

For servers that support relational databases, cleanup functions include:

- Terminating target SQLAM manager instances
- Performing required additional cleanup

3. Deallocates the SNA conversation.
4. Performs additional required cleanup so the SCM can attempt some form of recovery.

The SCM does the following when a communications failure occurs, and the UOW is not in the in doubt state:

1. Issues a BACKOUT verb to the local server.

For servers that support recoverable resources, this includes performing a backout on currently accessed recoverable resources.

2. Notifies the source agent that a communications failure has occurred.

Upon notification of the failure, the source agent notifies the other server managers to perform cleanup functions. These cleanup functions include:

- Performing required additional cleanup
- Notifying the requester that a communications failure has occurred
- Performing a backout on currently accessed recoverable resources

3. Deallocates the SNA LU 6.2 conversation that exists between the SCM and the TCM.

Reestablishment of the communications connectivity between the source and target systems is outside the scope of the DDM architecture. The SCM can reestablish communications with the TCM by allocating a new SNA LU 6.2 communications conversation. More information about initiating sync point communications is in the description of the SYNCMNI command.

### Protocol Flow Sequences

The following terms illustrate the SNA LU 6.2 protocols and the CMNSYNCPT conversational protocols used to implement this communications manager.

SYNCMNI	LU 6.2 Sync Point Communications Initiation (see <i>SYNCMNI</i> (on page 931))
SYNCMNT	LU 6.2 Sync Point Communications Termination (see <i>SYNCMNT</i> (on page 935))
APPSRCCD	LU 6.2 Source Command with Data (see <i>APPSRCCD</i> (on page 79))
APPSRCCR	LU 6.2 Source Command Returning Data (see <i>APPSRCCR</i> (on page 86))
APPSRCER	LU 6.2 Source Detected Error (see <i>APPSRCER</i> (on page 91))
APPTRGER	LU 6.2 Target Detected Error (see <i>APPTRGER</i> (on page 95))
SYNCMNFL	LU 6.2 Sync Point Communications Failure (see <i>SYNCMNFL</i> (on page 928))
SYNCMNCM	LU 6.2 Sync Point Communications Two-Phase Commit (see <i>SYNCMNCM</i> (on page 926))
SYNCMNBK	LU 6.2 Sync Point Communications Backout (see <i>SYNCMNBK</i> (on page 924))

**Transaction Program Names**

- 07F0F0F1
- All transaction program names beginning with 07F6
- A non-SNA-registered transaction program name the target system assigns to invoke the CMNSYNCPT

**Manager-Level Compatibility**

Table 3-7 illustrates the function of the CMNSYNCPT as it has grown and changed through the levels of the DDM architecture.

**Table 3-7 SNA LU 6.2 Sync Point Conversational Communications**

<b>DDM Levels</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
CMNSYNCPT				4	4
<i>Manager Dependencies</i>					
SYNCPTMGR				4	4

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>NIL</b>
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>
<b>mgrlvl</b>	4
<b>mgrdepls</b>	<b>MANAGER DEPENDENCY LIST</b>
X'14C0'	INSTANCE_OF SYNCPTMGR - Sync Point Manager MGRVLN 4 NOTE The CMNSYNCPT must interface with the sync point manager for coordinated sync point services.
<b>vldattls</b>	<b>VALID ATTRIBUTES</b>
X'0019'	INSTANCE_OF HELP - Help Text
X'1452'	INSTANCE_OF MGRNAM - Manager Name
X'0045'	INSTANCE_OF TITLE - Title

**SEE ALSO**

- insvar** *MGRLVL* (on page 506)
- mgrdepls** *SYNCPTMGR* (on page 939)
- Semantic** *ACCRDB* (on page 42)  
*AGENT* (on page 61)  
*CMNMGR* (on page 196)  
*CMNOVR* (on page 201)  
*CMNSYNCPT* (on page 202)  
*INHERITANCE* (on page 437)  
*RDBCMM* (on page 728)  
*RDBOVR* (on page 740)  
*RDBRLLBCK* (on page 745)  
*RPYDSS* (on page 766)

*RQSDSS* (on page 774)  
*SQLAM* (on page 847)  
*SUBSETS* (on page 902)  
*SYNCMNI* (on page 931)  
*SYNCPTMGR* (on page 939)

**NAME**

CMNTCPIP — TCP/IP Communication Manager

**DESCRIPTION (Semantic)**

- Dictionary** QDDBASD
- Codepoint** X'1474'
- Length** \*
- Class** CLASS
- Sprcls** CMNMGR - Communications Manager

TCP/IP Communications Manager (CMNTCPIP) describes the communications manager that supports protocols by using Transport Control Protocol/Internet Protocol (TCP/IP) local communications facilities.

DDM models the TCP/IP environment, including a communications manager. This manager is extendible to interoperate with other system services. For example, the communications manager can properly extract the authentication data and pass this information to the security manager. See *TCPIPOVR* (on page 1000) for an overview of TCP/IP.

DDM provides the following support for DDM conversational protocols over TCP/IP.

**TCP/IP Socket Calls Supported**

Table 3-8 summarizes the TCP/IP Socket Call interface used by the DDM communications manager. CMNTCPIP uses a TCP socket; it does not use a UDP socket.

**Table 3-8** TCP/IP Communications Manager (CMNTCPIP) Socket Calls

Socket Call	Local Support	Remote Support	Notes
ACCEPT	NO	YES	
BIND	YES	YES	
CLOSE	YES	NO	
CONNECT	YES	YES	
GETPEERNAME	YES	YES	
GETSOCKETNAME	YES	YES	
LISTEN	NO	YES	
READ	YES	YES	
SOCKET	YES	YES	
WRITE	YES	YES	

**Communications Manager Initiation**

The DDM source communications manager (SCM) operates like any other transaction program. That is, it uses the local TCP/IP communications facility to communicate with the target communications manager (TCM) on the remote system. For a sample protocol sequence that involves communications initiation, see *TCPCMNI* (on page 994). TCP/IP connections can be established as the result of a CONNECT call after a connection socket has been allocated by the SOCKET call. The initiation of the TCP/IP connection is made in response to an application program requesting access to remote data management services.

### DDM Conversational Protocol

To ensure the orderly exchange of requests and replies, the following DDM conversational protocol rules *must* be followed:

1. The first RQSDSS sent by the SCM must contain an EXCSAT command.
2. The SCM must not send an RPYDSS to the TCM.
3. The TCM must not send an RQSDSS to the SCM.
4. The SCM can send only one RQSDSS, RQSDSS chain, OBJDSS chained to a RQSDSS, or a mixture of RQSDSSs and OBJDSSs chained together before waiting for an RPYDSS or OBJDSS from the TCM.
5. The TCM can send only a single RPYDSS, RPYDSS chain, single OBJDSS, OBJDSS chain, or a mixture of RPYDSSs and OBJDSSs chained together before waiting for the next RQSDSS from the SCM.

### Basic Protocol Flow

When the user application initially requests remote data management services either through its local data manager (LDM), or directly to DDM, connectivity must first be established between the SCM and the TCM. A unique instance of a source agent and, possibly, a communications manager is created on the source system, a TCP/IP connection with the TCM is requested by the SCM of the local source communications facility. The TCP/IP connection is maintained throughout the life of the connection. In normal operation, only SCM may decide to close the connection. TCM may not initiate the CLOSE call unless SCM has done so or after the TCP/IP communication failed.

Normally, it is not necessary to establish more than one SCM/TCM connection for each user application. For example, if a requester on the source system wants to access three files on the target server, then only one connection is required between the SCM and TCM. However, a source system implementation is not required to use only one connection for this situation.

At the initialization stage, the SCM that initiates a TCP/IP connection is the first party to talk in the connection through the SCM.

The source agent can now request the SCM to send a chain of RQSDSSs and OBJDSSs through the connection to the target agent and wait for the target to reply for a chain of RPYDSSs and OBJDSSs.

When the target agent receives a command, it locates the target of each command and passes the command to the command target for execution. Replies returned to the target agent from the command target are sent to the TCM where they are queued. When all transactions are completed, TCM sends all replies to the SCM.

Depending upon the contents of the commands received from the sources, the target may return either reply messages (RM) or data or both to the source. Therefore, it is necessary to provide separate help text for a source sending a command with data and for a source sending a command without data. For a sample protocol sequence that involves sending a command that has an associated command data object and receiving a single reply message or reply data object in return, see *TCPSRCCD* (on page 1007). For a sample protocol sequence that involves sending a single command and receiving several reply data objects in return, see *TCPSRCCR* (on page 1010).

The basic pattern of the source agent sending commands and the target agent sending replies or data is repeated until the application program no longer needs remote data management services from the target system. Now, the source agent requests the SCM to terminate the

connection with the TCM. The SCM uses the local communications facility to terminate the connection with the TCM. The TCP/IP's CLOSE function terminates the SCM/TCM connection.

### **Normal Communication Termination**

By convention, only the SCM may terminate the connection between the SCM and the TCM in normal operation. The SCM should only terminate the connection when the source system has completed all of its work with the target system. For a sample protocol sequence that involves normal communications termination, see *TCPCMNT* (on page 997).

When the SCM deallocates its connection with the TCM, the TCM notifies the target agent so the agent can notify the other server managers to perform any required cleanup. For servers that support relational databases, this includes:

- Performing a rollback on currently accessed RDBs
- Destroying any SQLAM that is currently bound to the agent
- Performing any additional cleanup such as cleaning up internal tables or control blocks

The SCM terminates without waiting to see if the TCM terminates normally. Therefore, the TCM cannot send an error message if all of the work it is performing has not been completed.

### **Source Manager-Detected Error**

When the SCM processes the DSSs received from the TCM, an error may be detected by the SCM, the source agent, or other source manager that prevents the DSS contents from being processed. The error might be that the TCM sent structures that were not expected by the SCM. In this case, the SCM may decide to notify TCM of the error and close the connection. The TCM may not have to do anything with this error indication. It may wish to log this error or notify someone on the local system, but this is not a requirement. For a sample protocol sequence that involves a source manager-detected error, see *TCPSRCER* (on page 1013).

Recovery after such an error is the responsibility of the source agent. The source agent can attempt the command sequence again, return notification to the application program of the error and let it perform recovery, or have the SCM terminate the connection with the TCM based on the assumption that the TCM is inoperative. DDM does not have an architected recovery protocol.

### **Target Manager-Detected Error**

When the TCM processes the data structures received from the SCM, an error may be detected that prevents the TCM or target agent from processing the data structure. The error might be that the SCM sent structures that were not expected by the TCM. In this case, TCM will send a SYNTAXRM with error condition to SCM and let SCM close the connection. For a sample protocol sequence that involves a target manager-detected error, see *TCPTRGER* (on page 1015).

Recovery after such an error is the responsibility of the source agent. The source agent can attempt the command sequence again, return notification to the application program of the error and let it perform recovery, or request that the SCM terminate the connection with the TCM based on the assumption that the SCM is inoperative. DDM does not have an architected recovery protocol.

The TCM may wish to log the error or notify someone on the local system but this is not a requirement.

### Communication Failures

A communications failure can be caused by a TCP/IP connection protocol error, a connection outage, a communications line failure, a modem failure, a remote system failure, or failures of many other types. The result is that the SCM and TCM cannot communicate. Do not confuse communication failures with DDM-detected errors that result in a reply message. For a sample protocol sequence that involves a communications failure, see *TCPCMNFL* (on page 992).

The TCM does the following when a communications failure occurs:

1. Notifies the target agent that a communication failure has occurred. Upon notification of the failure, the target agent notifies the other server managers to perform required cleanup functions.

For servers that support files, this includes:

- Completing, if necessary, any DDM command processing
- Releasing all record and stream locks being held
- Closing any files that are open
- Releasing all file locks being held
- Performing additional required cleanup such as freeing up DCLFIL associations

For servers that support relational databases, this includes:

- Performing a rollback on currently accessed RDBs
- Destroying target SQLAM manager instances
- Performing required cleanup

2. Close TCP/IP connection.
3. Performs required cleanup so the SCM can reconnect and attempt some form of recovery.

The SCM does the following when a communications failure occurs:

1. Notifies the source agent that a communication failure has occurred. Upon notification of the failure, the source agent notifies the other server managers to perform cleanup functions. This includes:
  - Performing required cleanup
  - Notifying the requester that a communications failure has occurred
2. Close TCP/IP connection.

### Agent Queuing

The interface model between the communications manager and the agent is a CALL/RETURN interface and also a queuing interface. The requests of the managers are passed through the agent to the communications manager which assigns a request correlator to each request as it adds the DSS to its queue.

The communications support for TCP/IP can use full duplex operation, and can have multiple concurrent connections running between two TCP hosts. When command chaining is used, the target DDM server can begin executing the commands and returning data (data or reply messages) to the source server before the entire chain is received.

**Well-Known Port**

The well-known port is the socket on the server that accepts all the connection requests from the requester sockets. The well-known port is registered so its identity is known to provide a particular service. All connections to the server are initiated through this port. A connection request on the well-known port will cause TCP, through *ACCEPT* processing, to create a new socket, bind it to an available port, and associate this new socket and port to the connecting requester. This frees up the well-known port to receive other connection requests.

The well-known ports for the DDM architecture are shown in Table 3-9.

**Table 3-9** Well-Known Port and Symbolic Name Assignment

Port Number	Symbolic Name	Port Usage Explanation
446	DDM-RDB	Access relational databases using the DRDA source and target servers with the SQL application manager.
447	DDM-DFM	Access record-oriented files using the DDM source and target servers with the record and byte stream access methods.
448	DDM-SSL	Access relational databases using DRDA and/or DDM protocols along with Secure Sockets Layer (SSL) protocols to provide data encryption services.

**Protocol Flow Sequences**

The following terms illustrate the TCP/IP protocols and DDM conversational protocols used to implement this communication manager.

- TCPCMNI TCP/IP Communications Initiation (see *TCPCMNI* (on page 994))
- TCPCMNFL TCP/IP Communications Failure (see *TCPCMNFL* (on page 992))
- TCPCMNT TCP/IP Communications Termination (see *TCPCMNT* (on page 997))
- TCPSRCCD TCP/IP Source Command with Data (see *TCPSRCCD* (on page 1007))
- TCPSRCCR TCP/IP Source Command Returning Data (see *TCPSRCCR* (on page 1010))
- TCPSRCER TCP/IP Source Detected Error (see *TCPSRCER* (on page 1013))
- TCPTRGER TCP/IP Target Detected Error (see *TCPTRGER* (on page 1015))

<b>clsvar</b>	NIL
<b>insvar</b>	NIL
<b>clscmd</b>	NIL
<b>inscmd</b>	NIL
<b>mgrlvln</b>	5
<b>mgrdepls</b>	NIL
<b>vldattls</b>	VALID ATTRIBUTES



X'0019'	INSTANCE_OF	HELP - Help Text
X'1452'	INSTANCE_OF	MGRNAM - Manager Name
X'0045'	INSTANCE_OF	TITLE - Title

**SEE ALSO**

<b>insvar</b>	<i>MGRLVL</i> (on page 506) <i>SYNCLOG</i> (on page 922) <i>TCPPTHOST</i> (on page 1006)
<b>mgrdepls</b>	<i>RSYNCMGR</i> (on page 787) <i>SYNCPTMGR</i> (on page 939) <i>XAMGR</i> (on page 1063)
<b>Semantic</b>	<i>AGENT</i> (on page 61) <i>CMNMGR</i> (on page 196) <i>CMNOVR</i> (on page 201) <i>MGRLVLN</i> (on page 509) <i>RPYDSS</i> (on page 766) <i>RQSDSS</i> (on page 774) <i>RSYNCMGR</i> (on page 787) <i>SQLAM</i> (on page 847) <i>SUBSETS</i> (on page 902) <i>SYNCPTMGR</i> (on page 939) <i>TCPHOST</i> (on page 999) <i>TCPIPOVR</i> (on page 1000)

**NAME**

CNNTKN — Connection Token

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'1070'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Connection Token (CNNTKN) byte string specifies a token that is exchanged between a source and target SYNCPTMGR to identify an instance of a server. If multiple instances of a server have the same unit of work identifier (UOWID), this token is used to associate a resynchronization request to a server instance.

The CNNTKN is not to be considered as part of the log name. It is only used to uniquely identify a specific unit of work instance for a UOWID when the server system may have multiple server instances for the same UOWID.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	8	
class	X'1070'	
value	INSTANCE_OF	BYTSTRDR - Byte String
	LENGTH	4
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *SYNCLOG* (on page 922)

**Semantic** *SYNCLOG* (on page 922)  
*SYNCPTOV* (on page 944)

**NAME**

CNSVAL — Constant Value Attribute

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'000B'

**Length** \*

**Class** CLASS

**Sprcls** SCALAR - Scalar Object

Constant Value Attribute (CNSVAL) is a value that cannot be changed.

The instances of class CNSVAL specify a single constant value. The single value is specified as a literal of the required data class.

When CNSVAL is used as an attribute in the definition of a variable, the value of the CNSVAL attribute is the unchanging value of the variable.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>		<b>CLASS INSTANCE VARIABLES</b>
<b>length</b>	*	
<b>class</b>	X'000B'	
<b>value</b>	SPRCLS NOTE	FIELD - A Discrete Unit of Data The attributes for this variable vary according to what is being defined. Also, when BITDR, BITSTRDR, HEXDR, or HEXSTRDR values are specified, they are represented as being left-justified in the fewest number of whole bytes possible. For example, 123 would be X'1230'.
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

None.

## NAME

CNTQRY — Continue Query

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD

**Codepoint** X'2006'

**Length** \*

**Class** CLASS

**Sprcls** COMMAND - Command

The Continue Query (CNTQRY) command resumes a query or the return of result set data generated by the invocation of a stored procedure that the target SQLAM suspends.

The target SQLAM either terminates or suspends the query or result set at the completion of each OPNQRY, CNTQRY, or EXCSQLSTT command. The target SQLAM explicitly terminates a query or result set by returning a reply message (except OPNQRYRM, QRYPOPRM, or RSLSETRM) followed by an SQLCARD object. The target SQLAM suspends a query or result set based on the query protocols being used and the option in effect for the return of the EXTDTA object. For detailed information, see *EXTDTA* (on page 397), *FIXROWPRC* (on page 417), *LMTBLKPRC* (on page 475), *OUTOVR* (on page 572), *OUTOVROPT* (on page 576), and *RTNEXTDTA* (on page 792).

**Source System Processing**

The source system determines the location of the RDB:

**Local** Call the local RDB server.

**Remote** Send the CNTQRY command to the remote RDB server.

If there are LOBs in the output, the source system may send an OUTOVR object after the command if the application specified one and the requested *outovropt* allows for one to be sent. The OUTOVR object is required if the application wishes to receive LOB columns as LOB locator data and not as LOB data. Otherwise, the OUTOVR object is optional.

If the cursor is non-scrollable (the OPNQRYRM had a QRYATTSCR value of FALSE or the QRYATTSCR parameter was not specified on the OPNQRYRM), then the source SQLAM may specify QRYROWSET to cause a DRDA rowset to be returned for a cursor or result set conforming to the limited block query protocol. If the cursor is opened with rowset positioning (the OPNQRYRM had a QRYATTSET value of TRUE), then the source SQLAM must specify QRYROWSET to cause an SQL rowset to be returned for a cursor or result set conforming to the fixed row query protocol.

If an OPNQRY command contains a QRYROWSET parameter and the cursor is a non-scrollable cursor conforming to the limited block query protocol, then a QRYROWSET parameter must be present on every CNTQRY command for the cursor. Similarly, if the EXCSQLSTT command for a stored procedure call contains a QRYROWSET parameter, the QRYROWSET parameter applies only to a non-scrollable result set returned by the stored procedure if the result set conforms to the limited block query protocol and the QRYROWSET parameter must be present on every CNTQRY command for the result set. On the other hand, if the QRYROWSET parameter is not present on the OPNQRY or EXCSQLSTT command, it is not allowed on any CNTQRY command for such cursors. If the non-scrollable, non-rowset cursor or result set conforms to the fixed row query protocol, the QRYROWSET parameter is ignored on all such commands.

If the cursor is scrollable (the OPNQRYRM had a QRYATTSCR value of TRUE), then the source SQLAM specifies scrolling parameters on the CNTQRY request in accordance with the requirements of the application. If the source SQLAM does not support scrollable cursors, then it may use the opened cursor in a non-scrollable (serial) manner by omitting all scrolling parameters on any CNTQRY command.

If a QRYROWSET value is specified for a non-dynamic scrollable, non-rowset cursor, then the value indicates the number of rows in the DRDA rowset and the effect of the CNTQRY is equivalent to performing that number of single-row fetches across the network, but since they are retrieved with one command, the operation is more network-efficient. Since the target SQLAM is fetching rows in advance of the application's requesting those rows, the cursor position known to the application may be different from the cursor position known to the target SQLAM. It is the responsibility of the source SQLAM to manage such differences (see the DRDA Reference, Appendix B for an overview of scrollable cursors).

If a QRYROWSET value is specified for a rowset cursor, then the value indicates the number of rows in the rowset and the effect of the CNTQRY is equivalent to performing a multi-row fetch.

For a non-dynamic scrollable cursor, usage of the QRYROWSET parameter by the source SQLAM depends on both the QRYPRCTYP returned on the OPNQRYRM and the performance requirements of the source SQLAM.

If the QRYPRCTYP is FIXROWPRC, then each CNTQRY will execute at most one FETCH request if no QRYROWSET value is specified. Specification of a QRYROWSET value is required for rowset cursors and optional for non-rowset cursors. Generally, non-rowset cursors are used by the source SQLAM to enhance performance. It is possible to mix scrollable and non-scrollable behavior for these kinds of cursors by either specifying or omitting scrolling parameters on the CNTQRY command. As long as no QRYROWSET value is specified, there is no difference between the position of the cursor at the target SQLAM and at the source SQLAM.

If the QRYPRCTYP is LMTBLKPRC, then the presence of a QRYROWSET parameter on the OPNQRY command or on the first CNTQRY for the cursor determines whether the source SQLAM will be accessing the cursor in a scrollable manner using a non-rowset cursor. It is not possible to mix scrollable and non-scrollable behavior for these kinds of cursors:

- If the QRYROWSET parameter is not present on the OPNQRY command and on the first CNTQRY command, the source SQLAM will not be accessing the cursor in a scrollable manner and no scrolling parameters will be allowed on subsequent CNTQRY requests. Since the cursor is indeed scrollable, the source SQLAM must send a CLSQRY to close the cursor (since an SQLSTATE of 02000 does not close the cursor as it does for non-scrollable cursors), if the cursor was not previously closed (for example, by a commit, rollback, or a terminating error).
- If the QRYROWSET parameter is present on the first CNTQRY, the source SQLAM will be accessing the cursor in a scrollable manner and the QRYROWSET parameter is required on all subsequent CNTQRY requests, even if the value is 1. A QRYROWSET value greater than one is used to enhance performance. If QRYROWSET is on the OPNQRY command, then it is required on all CNTQRY commands. If QRYROWSET is not on the OPNQRY, then it is required on the first CNTQRY command to indicate that the cursor will be used in a scrollable manner.

If the source SQLAM did not specify a QRYROWSET parameter on the OPNQRY command for a scrollable cursor and then subsequently encounters a partial row in the QRYDTA returned, then the QRYPRCTYP parameter returned on the OPNQRYRM must have been LMTBLKPRC and the QRYBLKTYP must have been QRYBLKEXA (exact blocking). The rows returned are members of an implicit rowset of a size determined by Query Data Transfer Protocol rule QP4. If

the source SQLAM will access the cursor in a scrollable manner, then the DRDA rowset must either be completed or reset. If the source SQLAM will not access the cursor in a scrollable manner, then this is not required.

### Data Conversion

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

### DSS Carrier: RQSDSS

### Target System Processing

The optional *cmdsrcid* parameter uniquely identifies the source of the query.

The *maxblkext* parameter specifies the maximum number of extra blocks of reply data objects and reply messages that the requester is capable of receiving in response to the CNTQRY command.

The *monitor* parameter allows the requester to request a set of events or activities to be monitored on the target server. Refer to MONITOR for the list of supported events.

The *pkgnamcsn* parameter specifies the fully qualified package name, the package consistency token, and a section number within the package. The *pkgnamcsn* parameter must match the *pkgnamcsn* parameter of the OPNQRY command that opened the query.

The *qryblkst* parameter specifies whether any pending partial row or pending query blocks (QRYDTA and EXTDTA) are to be discarded. This operation is used in conjunction with scrollable cursors to terminate an incomplete DRDA rowset.

The *qryblksz* parameter specifies the query block size for the reply data objects and reply messages to resume the query. The value of the *qryblksz* parameter can be different than the value specified on the OPNQRY command, EXCSQLSTT command, or any previous CNTQRY command. The target SQLAM must conform to the specified query block size. This parameter is ignored for rowset cursors which are not blocked.

The *qryinsid* parameter specifies the instance of the query in use. Its value must match the *qryinsid* parameter returned on the OPNQRYRM reply message for this instance of the query.

The *qryrownbr* parameter specifies which row (absolute) or how many rows from the current row (relative) to begin fetching in a scrollable cursor. It is used in conjunction with *qryscorn* to perform scrolling operations on the cursor. For absolute positioning, the first row is +1, the last rowset ends at -1. For relative positioning, +1 is the next row, -1 is the previous row, and so on.

The *qryrownbr* parameter for non-rowset cursors specifies which row (absolute) or how many rows from the current rowset (relative) to begin fetching in a scrollable rowset cursor. It is used in conjunction with *qryscorn* to perform scrolling operations on the cursor. For absolute positioning, the first rowset starts at +1, the last rowset starts at -1. For relative positioning, assuming *k* is equal to the number of rows for the current rowset, then +*k* is the next rowset, -*k* is the previous rowset, and so on.

The *qryrowset* parameter specifies whether a rowset of rows is to be returned with the command. This is allowed for all rowset cursors, for non-dynamic scrollable non-rowset cursors, and for non-scrollable non-rowset cursors conforming to the limited block query protocol. The target server fetches no more than the requested number of rows. It may fetch fewer rows if it is restricted by extra query block limits, or if a fetch operation results in a negative SQLSTATE or

an SQLSTATE of 02000, or if the source SQLAM requested a positioning fetch.

The *qryrowsns* parameter specifies whether the rows to be fetched from a scrollable cursor are to be sensitive to changes in the data underlying the rows in the result table.

The *qyrtdta* parameter specifies whether data is to be returned with the command. If FALSE is sent on the command, the result is that only a positioning fetch operation is performed.

The *qyrcrorn* parameter controls the scroll orientation of a scrollable cursor. It is used in conjunction with the *qyrownbr* to perform scrolling operations on the cursor.

The *rdbnam* parameter specifies the name of the relational database (RDB) that the ACCRDB command accesses. If the *rdbnam* parameter is specified, its value must be the same as the *rdbnam* parameter on the ACCRDB command.

The CNTQRY command is valid any time that the target SQLAM suspends a query or result set.

If the OPNQRY or EXCSQLSTT command or a previous CNTQRY command saved a reply message and SQLCARD object, then the saved reply message and SQLCARD object are returned, and the query or result set is terminated.

The SECTKNOVR reply data object is sent by the intermediate server, if QRYDTA and/or EXTDTA reply data objects are encrypted. The SECTKNOVR contains the encryption seed and the encryption token used to encrypt the QRYDTA and EXTDTA reply data objects. The SECTKNOVR DSS is encrypted and must be sent prior to the first encrypted reply data object.

The TYPDEFNAM reply data object specifies the name of the data type to data representation mapping definitions that the target SQLAM uses when sending the SQLCARD object for this command.

The TYPDEFOVR reply data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type to data representation mapping definitions for the SQLCARD object for this command.

If there are LOBs in the output, the following applies.

For LMTBLKPRC queries or result sets, the *rtnextdta* parameter sent by the source on the CNTQRY command specifies the manner in which the target is to return EXTDTA objects to the source system if any are part of the query answer set or result set. If a *qryrowset* value is specified on the CNTQRY, then the *rtnextdta* parameter must be set to *rtnextall*. See *QRYROWSET* (on page 697) for details.

The *outovrany* option specified on the OPNQRY or EXCSQLSTT command, indicates when the application requester will send an OUTOVR object. The OUTOVR object is sent by the application requester to determine the format to be used by the application server in returning LOB data columns to the application requester.

If the *outovrfrs* option is specified, then only the OUTOVR object sent with the first CNTQRY command is used to determine the format to be used in returning LOB data columns to the application requester. All subsequent OUTOVR objects are rejected with an OBJNSPRM.

If an OUTOVR object is not sent with a CNTQRY command, then the data is returned in the same format as specified in the last CNTQRY command if there was one. If the CNTQRY command is the first one, then the data is returned in the format of the data given in the QRYDSC object.

Normal completion of the CNTQRY command suspends the query and returns zero or more QRYDTA objects and zero or more EXTDTAs.

An RDBUPDRM reply message may be returned in the reply chain in accordance with the DRDA Update Control (UP) rules (refer to the DRDA Reference for details).

### Exceptions

Exception conditions that the RDB detects are reported in the SQLCARD or QRYDTA reply data objects. If a QRYDTA reply data object is returned and FIXROWPRC query protocols are used with a single row fetch, then any reply message and following SQLCARD must be saved and not returned in response to the CNTQRY. They are returned in response to the next CNTQRY command. In the case of LMTBLKPRC query protocols and FIXROWPRC query protocols with a blocked fetch, the reply message and SQLCARD are returned according to the DRDA Query Data or Result Set Termination rules (QT) that apply for the SQLAM level in effect (see the DRDA Reference for further information on these rules).

For non-scrollable cursors, an ENDQRYRM and an SQLCARD object are returned after any QRYDTA reply data objects, and the query is terminated if all the following conditions are met:

- The last of the answer set data is returned (SQLSTATE 02000), and the target server has determined it can close the query implicitly based on the cursor type and the QRYCLSIMP value as specified previously on the OPNQRY command, or some RDB-detected error condition has occurred.
- LMTBLKPRC query protocols or FIXROWPRC query protocols with a block fetch are used.
- The DRDA Query Termination rules (QT) in effect for the SQLAM level do not prohibit the ENDQRYRM and SQLCARD.

For scrollable cursors, the query is not terminated when the last of the answer set data is returned. An RDB-detected error condition can terminate the query and results in the return of the ENDQRYRM and SQLCARD. Otherwise, a CLSQRY command from the source SQLAM is required to close the cursor if the cursor was not previously closed (for example, by a commit, rollback, or a terminating error).

If the ENDQRYRM and SQLCARD are not returned with the CNTQRY reply, they are retained by the target server for return with a later command and the query or result set is suspended.

If not enough space is available in the current query block for the ENDQRYRM and the SQLCARD object, then the query or result set is suspended.

If the bind process is active, then the command is rejected with the PKGBPARAM, and the state of the query is not changed.

If the query or result set is not open, then the command is rejected with the QRYNOPRM.

If the specified RDB is not currently accessed, the command is rejected with the RDBNACRM.

If the current unit of work is rolled back (possibly due to a deadlock situation) and a QRYDTA reply object is returned, then the following should occur: if an SQLAM issues the command, then an ABNUOWRM and an SQLCARD object must be saved and not returned in response to the CNTQRY. They are returned in response to the next command.

For rowset cursors, an SQLCARD object is returned when:

- The QRYDTA contains the last row of a rowset but the cursor is still open, or
- The QRYDTA contains a cursor ending condition where the SQLCARD is returned with the ENDQRYRM.



**Source System Reply Processing**

Return any reply messages and data objects to the requester.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**Package Name and Consistency Token**

With SQLAM level 7, the PKGNAMCSN is changed to an optional instance variable. If specified, the values are used as the default package name and consistency token for the connection. If not specified on the command, the PKGSN is required to identify the section number. In this case, the default package name, consistency token and specified section number is used for the request. If PKGNAMCSN was never specified on the connection, a PRCCNVRM is returned.

clsvar	NIL	
insvar	CLASS INSTANCE VARIABLES	
length	*	
class	X'2006'	
cmdsrcid	INSTANCE_OF OPTIONAL MINLVL	CMDSRCID - Command Source Identifier 7
maxblkext	INSTANCE_OF OPTIONAL DFTVAL	MAXBLKEXT - Maximum Number of Extra Blocks 0
monitor	INSTANCE_OF OPTIONAL MINLVL	MONITOR - Monitor Events 7
pkgnamcsn	INSTANCE_OF  MINLVL OPTIONAL MTLEXC	PKGNAMCSN - RDB Package Name, Consistency Token, and Section Number 7 Required if PKGSN not specified PKGSN
pkgsn	INSTANCE_OF MINLVL OPTIONAL MTLEXC	PKGSN - RDB Package Section Number 7 Required if PKGNAMCSN not specified. PKGNAMCSN
qryblkrst	INSTANCE_OF OPTIONAL MINLVL	QRYBLKRST - Query Block Reset 7
qryblksz	INSTANCE_OF REQUIRED IGNORED	QRYBLKSZ - Query Block Size  If CNTQRY is FIXROWPRC and the OPNQRYRM specified QRYATTSET indicating a rowset cursor.
qryinsid	INSTANCE_OF	QRYINSID - Query Instance Identifier

	REQUIRED MINLVL	7
qryrownbr	INSTANCE_OF REQUIRED IGNORED MINLVL	QRYROWNBR - Query Row Number If QRYSCRORN is QRYSCRREL or QRYSCRABS If QRYSCRORN is QRYSCRAFT or QRYSCRBEF 7
qryrowset	INSTANCE_OF REQUIRED  REQUIRED  REQUIRED  NOT ALLOWED  NOT ALLOWED  OPTIONAL MINLVL	QRYROWSET - Query Rowset Size If CNTQRY is FIXROWPRC and the OPNQRYM specified QRYATTSET indicating a rowset cursor. If QRYPRCTYP is LMTBLKPRC and the OPNQRY command or the first CNTQRY for a non-dynamic scrollable query specified a QRYROWSET. If QRYPRCTYP is LMTBLKPRC and the OPNQRY command for a non-scrollable query specified a QRYROWSET. If QRYPRCTYP is LMTBLKPRC and the first CNTQRY for a non-dynamic scrollable query did not specify QRYROWSET. If QRYPRCTYP is LMTBLKPRC and the OPNQRY for a non-scrollable query did not specify QRYROWSET. Otherwise. 7
qryrowsns	INSTANCE_OF OPTIONAL MINLVL	QRYROWSNS - Query Row Sensitivity 7
qyrtnmdta	INSTANCE_OF OPTIONAL IGNORED MINLVL	QRYRTNMDTA - Query Returns Data If QRYSCRORN is QRYSCRAFT 7
qryscrom	INSTANCE_OF OPTIONAL MINLVL	QRYSCRORN - Query Scroll Orientation 7
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
rtnextdta	INSTANCE_OF OPTIONAL NOTE	RTNEXTDTA - Return of EXTDTA Option Ignored if query is not LMTBLKPRC.
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>cmddta</b>		<b>COMMAND OBJECTS</b>
X'2415'	INSTANCE_OF OPTIONAL NOTE	OUTOVR - Output Override Descriptor Specified if source system is overriding the format of the output host variables from either a previous CNTQRY command or the QRYDSC returned with the OPNQRYM when the query was opened.

<b>rpydta</b>		<b>REPLY OBJECTS</b>
X'002F'	INSTANCE_OF	TYPDEFNAM - Data Type Definition Name
	ENUVAL	'QTDSQL370'
	ENUVAL	'QTDSQL400'
	ENUVAL	'QTDSQLX86'
	ENUVAL	'QTDSQLASC'
	MINLVL	4
	ENUVAL	'QTDSQLVAX'
	MINLVL	4
	OPTIONAL	
	DFTVAL	''
	NOTE	The default means the value received on ACCRDBRM is used. If sent, the value does not apply to the data in QRYDTA or the EXTDTA objects.
NOTE	Ignored if sent with EXTDTAs.	
X'0035'	INSTANCE_OF	TYPDEFOVR - TYPDEF Overrides
	OPTIONAL	
	DFTVAL	''
	NOTE	The default means the value received on ACCRDBRM is used.
NOTE	If sent, the value does not apply to the data in QRYDTA objects or EXTDTAs.	
NOTE	Ignored if sent with EXTDTAs.	
X'190B	INSTANCE_OF	SECTKNOVR - SECTKN Override
	OPTIONAL	
	NOTE	This is sent by the intermediate server, if QRYDTA and/or EXTDTA are encrypted. This must precede the first encrypted reply data object.
X'1C00'	INSTANCE_OF	MONITORRD - Monitor Reply Data
	OPTIONAL	
	MINLVL	7
X'1C03'	INSTANCE_OF	MGRLVLOVR - Manager Level Overrides
	OPTIONAL	
	MINLVL	7
X'2408'	INSTANCE_OF	SQLCARD - SQL Communications
	OPTIONAL	Area Reply Data
	NOTE	The SQLCARD object cannot be returned without also returning a reply message. The reply message returned with the SQLCARD always precedes the SQLCARD object.
X'241B'	INSTANCE_OF	QRYDTA - Query Answer Set Data
	OPTIONAL	
	REPEATABLE	
	NOTE	At least one of QRYDTA or one EXTDTA must be returned, but QRYDTA(s) may be returned without any EXTDTAs and EXTDTA(s) may be returned without any QRYDTAs.

X'146C'	INSTANCE_OF OPTIONAL REPEATABLE NOTE	EXTDTA - Externalized FD:OCA Data  At least one of QRYDTA or one EXTDTA must be returned, but QRYDTA(s) may be returned without any EXTDTAs and EXTDTA(s) may be returned without any QRYDTAs.
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'220D'	INSTANCE_OF NOTE	ABNUOWRM - Abnormal End Unit of Work Condition Returned only if an SQLAM issues the command.
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported
X'220B'	INSTANCE_OF	ENDQRYRM - End of Query
X'2209'	INSTANCE_OF	PKGBPARM - RDB Package Binding Process Active
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'2202'	INSTANCE_OF	QRYNOPRM - Query Not Open
X'2204'	INSTANCE_OF	RDBNACRM - RDB Not Accessed
X'2218'	INSTANCE_OF	RDBUPDRM - RDB Update Reply Message
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limits Reached
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'125F'	INSTANCE_OF	TRGNSPRM - Target Not Supported
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

## SEE ALSO

**insvar**

- EXTDTA* (on page 397)
- LMTBLKPRC* (on page 475)
- MAXBLKEXT* (on page 494)
- MONITOR* (on page 521)
- NBRROW* (on page 530)
- OPNQRY* (on page 555)
- OUTOVROPT* (on page 576)
- PKGNAMECSN* (on page 596)
- PKGSN* (on page 606)
- QRYBLKRST* (on page 677)
- QRYBLKSZ* (on page 678)
- QRYINSID* (on page 688)
- QRYPRCTYP* (on page 692)
- QRYROWNBR* (on page 694)
- QRYROWSET* (on page 697)
- QRYROWSNS* (on page 701)
- QRYSCRABS* (on page 703)
- QRYSCRAFT* (on page 704)
- QRYSCRORN* (on page 710)
- QRYSCRREL* (on page 713)
- RDBNAM* (on page 736)

	<i>RTNEXTDTA</i> (on page 792)
<b>inscmd</b>	<i>SQLAM</i> (on page 847)
<b>cmddta</b>	<i>OPNQRYRM</i> (on page 566) <i>OUTOVR</i> (on page 572) <i>QRYDSC</i> (on page 685)
<b>rpydta</b>	<i>ACCRDBRM</i> (on page 48) <i>EXTDTA</i> (on page 397) <i>MONITORRD</i> (on page 523) <i>QRYDTA</i> (on page 686) <i>SQLCARD</i> (on page 855) <i>TYPDEFNAM</i> (on page 1027)
<b>cmdrpy</b>	<i>ABNUOWRM</i> (on page 39) <i>AGNPRMRM</i> (on page 65) <i>CMDATHRM</i> (on page 171) <i>CMDCHKRM</i> (on page 173) <i>CMDNSPRM</i> (on page 176) <i>ENDQRYRM</i> (on page 342) <i>PKGBPARM</i> (on page 586) <i>PRCCNVRM</i> (on page 625) <i>PRMNSPRM</i> (on page 633) <i>QRYNOPRM</i> (on page 690) <i>RDBNACRM</i> (on page 734) <i>RSCLMTRM</i> (on page 778) <i>SYNTAXRM</i> (on page 989) <i>TRGNSPRM</i> (on page 1022) <i>VALNSPRM</i> (on page 1057)
<b>Semantic</b>	<i>ABNUOWRM</i> (on page 39) <i>ACCRDB</i> (on page 42) <i>APPSRCCR</i> (on page 86) <i>BGNBND</i> (on page 110) <i>CCSIDMGR</i> (on page 150) <i>CLASS</i> (on page 158) <i>CLSQR</i> Y (on page 165) <i>CMDSRCID</i> (on page 178) <i>COMMAND</i> (on page 240) <i>ENDQRYRM</i> (on page 342) <i>EXCSQLSTT</i> (on page 381) <i>EXTDTA</i> (on page 397) <i>FIXROWPRC</i> (on page 417) <i>LMTBLKPRC</i> (on page 475) <i>MAXBLKEXT</i> (on page 494) <i>MONITOR</i> (on page 521) <i>NBRROW</i> (on page 530) <i>OBJNSPRM</i> (on page 542) <i>OPNQRY</i> (on page 555) <i>OPNQRYRM</i> (on page 566) <i>OUTOVR</i> (on page 572) <i>OUTOVRANY</i> (on page 574) <i>OUTOVRFRS</i> (on page 575) <i>OUTOVROPT</i> (on page 576)

*PKGBPARAM* (on page 586)  
*PKGNAMECSN* (on page 596)  
*PKGSN* (on page 606)  
*PRCCNVRM* (on page 625)  
*QDDRDBD* (on page 657)  
*QRYATTSCR* (on page 665)  
*QRYBLK* (on page 670)  
*QRYBLKSZ* (on page 678)  
*QRYCLSIMP* (on page 680)  
*QRYDTA* (on page 686)  
*QRYDSC* (on page 685)  
*QRYINSID* (on page 688)  
*QRYNOPRM* (on page 690)  
*QRYOPRM* (on page 691)  
*QRYPRCTYP* (on page 692)  
*QRYROWNBR* (on page 694)  
*QRYROWSET* (on page 697)  
*QRYRTNDTA* (on page 702)  
*QRYSCRCUR* (on page 706)  
*QRYSCRFST* (on page 707)  
*QRYSCRLST* (on page 708)  
*QRYSCRNXT* (on page 709)  
*QRYSCRPRI* (on page 712)  
*RDBNACRM* (on page 734)  
*RSLSETRM* (on page 785)  
*RTNEXTALL* (on page 791)  
*RTNEXTDTA* (on page 792)  
*RTNEXTROW* (on page 793)  
*SECTKNOVR* (on page 822)  
*SQL* (on page 835)  
*SQLAM* (on page 847)  
*TYPDEFNAM* (on page 1027)  
*TYPDEFOVR* (on page 1030)

**NAME**

CODPNT — Codepoint

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'000C'

**Length** \*

**Class** CLASS

**Sprcls** HELP - Help Text

The Codepoint (CODPNT) Data specifies a scalar value that is an architected codepoint.

See *CODPNTDR* (on page 235) for information on the representation and use of codepoint data.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'000C'	
status	INSTANCE_OF TITLE	STSLST - Term Status Collection term status
title	INSTANCE_OF	TITLE - Title
<b>semantic</b>	INSTANCE_OF MAXLEN REPEATABLE TITLE	TEXT - Text Character String 80 description
codpnt	INSTANCE_OF	CODPNTDR - Codepoint Data Representation
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *CMDNSPRM* (on page 176)  
*DCTINDEN* (on page 259)  
*OBJNSPRM* (on page 542)  
*PRMNSPRM* (on page 633)  
*SYNTAXRM* (on page 989)  
*VALNSPRM* (on page 1057)

**Semantic** *CLASS* (on page 158)  
*BGNATMCHN* (on page 107)  
*DFTDATFMT* (on page 272)  
*DFTTIMFMT* (on page 275)  
*DMYBLKDATFMT* (on page 288)  
*DMYCMADATFMT* (on page 289)  
*DMYHPNDATFMT* (on page 290)  
*DMYPRDDATFMT* (on page 291)  
*DMYSLHDATFMT* (on page 292)  
*ENDATMCHN* (on page 333)  
*EXPALL* (on page 393)  
*EXPNON* (on page 394)  
*EXPREOPT* (on page 395)

*EXPYES* (on page 396)  
*FIXROWPRC* (on page 417)  
*HMSBLKTIMFMT* (on page 431)  
*HMSCLNTIMFMT* (on page 432)  
*HMSCMATIMFMT* (on page 433)  
*HMSPRDTIMFMT* (on page 434)  
*JULBLKDATFMT* (on page 460)  
*JULCMADATFMT* (on page 461)  
*JULHPNDATFMT* (on page 462)  
*JULPRDDATFMT* (on page 463)  
*JULSLHDATFMT* (on page 464)  
*LMTBLKPRC* (on page 475)  
*LOCDATFMT* (on page 482)  
*LOCTIMFMT* (on page 483)  
*MDYBLKDATFMT* (on page 500)  
*MDYCMADATFMT* (on page 501)  
*MDYHPNDATFMT* (on page 502)  
*MDYPRDDATFMT* (on page 503)  
*MDYSLHDATFMT* (on page 504)  
*PKTOBJ* (on page 608)  
*SNDPKT* (on page 829)  
*YMDBLKDATFMT* (on page 1113)  
*YMDCMADATFMT* (on page 1114)  
*YMDHPNDATFMT* (on page 1115)  
*YMDPRDDATFMT* (on page 1116)  
*YMDSLHDATFMT* (on page 1117)

**sprcls**      *CLASS* (on page 158)



**NAME**

CODPNTDR — Codepoint Data Representation

**DESCRIPTION (Semantic)****Dictionary** QDDPRMD**Codepoint** X'0064'**Length** \***Class** CLASS**Sprcls** FIELD - A Discrete Unit of Data

Codepoint Data Representation (CODPNTDR) specifies the data representation of a dictionary codepoint. Codepoints are hexadecimal aliases for DDM architecture's named terms. Codepoints reduce the number of bytes required to identify the class of an object in memory and in data streams.

Figure 3-18 (on page 236) illustrates how the codepoints of objects stored in memory or received from a data stream refer to the class descriptions stored in a dictionary for those objects.

Codepoints consist of two values, a 1-hex index into a list of dictionaries and a 3-hex identifier unique within the referenced dictionary. The resolution of codepoints to class descriptors is illustrated in Figure 3-18 (on page 236).

For Agent Level 3, the list of dictionaries is fixed, and specific index values are assigned as follows:

- C-F: Reserved for product extensions (see *EXTENSIONS* (on page 400))
- 4-B: Reserved for DDM Architecture
- 3: DDM dictionary QDDADLD
- 2: DDM dictionary QDDRDBD
- 1: DDM dictionary QDDBASD
- 0: DDM dictionary QDDPRMD

The assignment of codepoints within DDM dictionary QDDBASD is:

- 10\_\_: commands
- 11\_\_: parameters
- 12\_\_: reply messages
- 14\_\_: other classes of objects

Similar assignments are also made in QDDRDBD. This is not an architected standard or requirement.

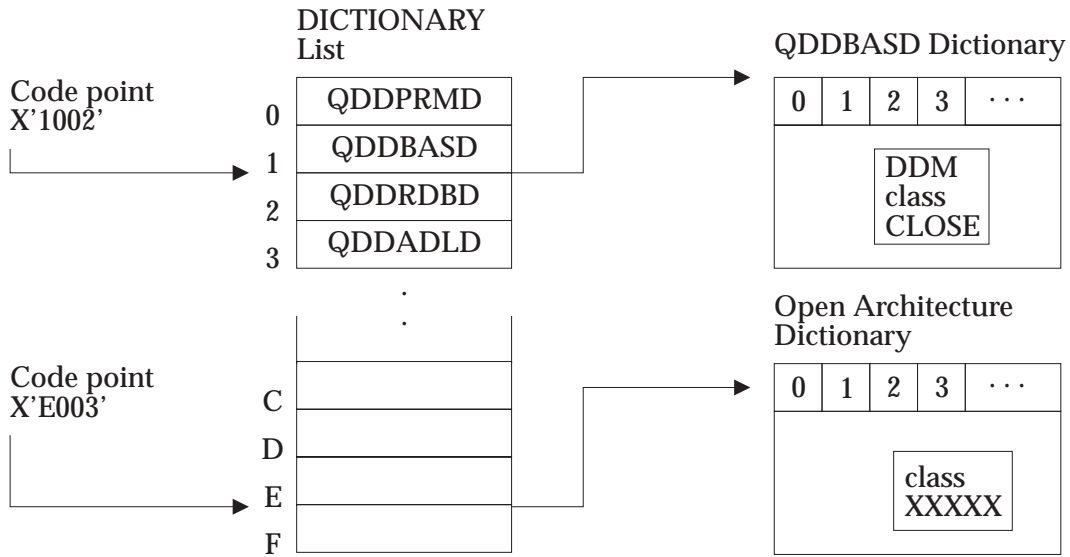


Figure 3-18 Codepoint References to Dictionaries

**Length Specification**

The length of CODPNTDR fields is two bytes.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>		<b>CLASS INSTANCE VARIABLES</b>
identifier	INSTANCE_OF LENGTH NOTE	HEXSTRDR - Hexadecimal String 3 Specifies a unique identifier associated with a specific term in a dictionary.
index	INSTANCE_OF NOTE	HEXDR - Hexadecimal Number Specifies an index into the dictionary list of the agent.
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>vldattls</b>	<b>NIL</b>	

**SEE ALSO**

- clsvar**            *QRYPRCTYP* (on page 692)
- insvar**            *BNDCHKEXS* (on page 125)
- BNDCRTCTL* (on page 127)
- BNDEXPOPT* (on page 129)
- BNDSTTASM* (on page 141)
- ELMCLS* (on page 330)
- ENUCLS* (on page 345)
- INSTANCE\_OF* (on page 444)
- MGRLVL* (on page 506)
- MTLEXC* (on page 524)
- MTLINC* (on page 525)

	<i>OBJECT</i> (on page 540)
	<i>PKGATHOPT</i> (on page 580)
	<i>PKGDFTCST</i> (on page 590)
	<i>PKGISOLVL</i> (on page 592)
	<i>PKGRPLOPT</i> (on page 604)
	<i>QRYBLKCTL</i> (on page 672)
	<i>RDBACCCL</i> (on page 723)
	<i>RDBRLOPT</i> (on page 750)
	<i>RSCTYP</i> (on page 782)
	<i>SPRCLS</i> (on page 832)
	<i>STTDATFMT</i> (on page 894)
	<i>STTDECDEL</i> (on page 897)
	<i>STTSTRDEL</i> (on page 899)
	<i>STTTIMFMT</i> (on page 900)
<b>Semantic</b>	<i>CODPNT</i> (on page 233)
	<i>EXTENSIONS</i> (on page 400)
<b>semantic</b>	<i>CODPNT</i> (on page 233)

**NAME**

COLLECTION — Collection Object

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

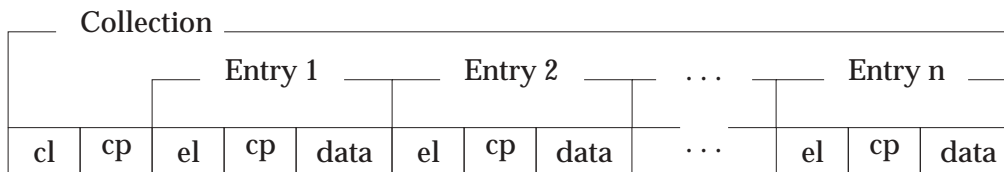
**Codepoint** X'000D'

**Length** \*

**Class** CLASS

**Sprcls** OBJECT - Self-identifying Data

Collection Object (COLLECTION) describes the abstract class of all collections of objects.



**Figure 3-19** Collection-to-Data-Stream Mapping

**Legend**

cl Length of collection.

cp A codepoint.

el Entry length.

data Entry data.

All objects in DDM are modeled as either scalars (see *SCALAR* (on page 796)) or collections. Collections are modeled as a structure with the following fields:

**Length** The total length of the collection, including the length and codepoint fields (4 bytes), and the length of all entries in the collection.

See *DSS* (on page 308) for a discussion of:

- Object segmentation in the data stream

- Objects with the data longer than 32 Kbytes less 5 bytes or 32763 bytes

**Code Point** A unique hexadecimal identifier assigned to each instance of a collection

**Entries** The objects within the domain of the collection

**Literal Form**

The literal form of all collections is:

```
collection_class_name(value...)
```

in which the collection values are specified as literals of the classes specified for the collection's instance variables.

For example, the literal form of the command to declare a file is:

DCLFIL (FILNAM (PHONELIST) DCLNAM (TELE) )

**Note:** In class descriptions of collection terms, the length and codepoint of the collection are always specified as data fields. They are then followed by the objects of the collection. If the length specified for the term is \*, the total length of all objects +4 bytes must be substituted for the \*.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
<b>length</b>	<b>4</b>
<b>class</b>	<b>X'000D'</b>
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

#### SEE ALSO

**Semantic**

- ACCSECRD* (on page 58)
- ARRAY* (on page 101)
- ATTLIST* (on page 105)
- BNDOPT* (on page 133)
- CLASS* (on page 158)
- COMMAND* (on page 240)
- DSS* (on page 308)
- EXCSATRD* (on page 369)
- INHERITANCE* (on page 437)
- OBJECT* (on page 540)
- OBJOVR* (on page 544)
- OOPOVR* (on page 547)
- ORDCOL* (on page 569)
- PKGDFTCC* (on page 589)
- PRPHRCLST* (on page 635)
- RPYMSG* (on page 770)
- SRVLST* (on page 878)
- SYNCCRD* (on page 913)
- SYNCRRD* (on page 981)
- TYPDEFOVR* (on page 1030)

## NAME

COMMAND — Command

## DESCRIPTION (Semantic)

**Dictionary** QDDPRMD**Codepoint** X'000E'**Length** \***Class** CLASS**Sprcls** COLLECTION - Collection Object

Command (COMMAND) Collection Object is a message sent to an object that requests a function. For example, the command *get record* can be sent to an access method.

Each DDM command normally returns one or more reply messages or data objects.

DDM commands are described in five parts:

1. Description:
  - Command name
  - Expanded name
  - Semantic (description of the command function)
2. Parameters:
  - The instance variables describe the objects that can (or must be) sent as parameters of the command.
  - The length and class of the command must be sent first and in the respective order. The rest of the instance variables are the parameters of the command.
  - The parameters can be sent in any order because they are identified by their class codepoints.
  - One and only one parameter of the command must have the *CMDTRG* (command target) attribute. This parameter identifies the object that is to receive and process the command. See *AGNCMDPR* (on page 64) and *CMDTRG* (on page 179) for a discussion of command targets.
3. Command data:
  - Lists all of the possible classes of data objects (for example, records) that can be associated with the command.
  - Each data object will have the attribute of *REQUIRED* or *OPTIONAL* with it. *REQUIRED* means that the data object will always be sent with the command. *OPTIONAL* means that the data object could be sent with the command, but it is not imperative.
4. Reply data:
  - Lists all possible classes of data objects that can be returned for the command.
  - The list may contain notes about selecting the data objects to return.
  - The reply data objects are normally returned for the command. When exception conditions occur, the reply data objects might not be returned, instead reply messages may return a description of the exception conditions.

5. Reply messages:

- List of all of the possible reply messages that can be returned for the command, including all severity code values.
- A target system can return as many of the possible reply messages as it supports. Some systems do not have or support the exception condition. For example, a system without security cannot return authorization reply messages.
- Reply messages can be returned instead of reply data objects.

The DDM communications manager maps all DDM commands onto the RQSDSS for transmission:

```
RQSDSS(command(command parameters))
```

The DDM communications manager maps all DDM command data objects and reply data objects onto an OBJDSS structure for transmission:

```
OBJDSS(command-data-object(object parameters))
OBJDSS(reply-data-object(object parameters))
```

The DDM communications manager maps all DDM command replies onto an RPYDSS structure for transmission:

```
RPYDSS(command-reply(reply parameters))
```

<b>clsvar</b>		<b>CLASS VARIABLES</b>
<b>cmddta</b>	INSTANCE_OF TITLE NOTE	DEFLST - Definition List command data objects All commands can specify the data objects to be transmitted in OBJDSSs.
<b>rpydta</b>	INSTANCE_OF TITLE NOTE	DEFLST - Definition List reply data objects All commands can specify the data objects to be returned to the requester in OBJDSSs.
<b>cmdrpy</b>	INSTANCE_OF TITLE NOTE	DEFLST - Definition List command replies All commands can specify the reply messages that can be generated for them.
<b>insvar</b>		<b>CLASS INSTANCE VARIABLES</b>
length	*	
class	X'000E'	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	
<b>cmddta</b>	NIL	
<b>rpydta</b>	NIL	
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command

X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limits Reached
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'125F'	INSTANCE_OF	TRGNSPRM - Target Not Supported
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

**SEE ALSO**

<b>Semantic</b>	<i>ACCRDB</i> (on page 42)
	<i>ACCSEC</i> (on page 52)
	<i>BGNATMCHN</i> (on page 107)
	<i>BGNBND</i> (on page 110)
	<i>BNDSQLSTT</i> (on page 136)
	<i>CLSQR</i> Y (on page 165)
	<i>CNTQR</i> Y (on page 222)
	<i>DRPPKG</i> (on page 293)
	<i>DSCRDBTBL</i> (on page 300)
	<i>DSCSQLSTT</i> (on page 304)
	<i>DSS</i> (on page 308)
	<i>ENDATMCHN</i> (on page 333)
	<i>ENDBND</i> (on page 336)
	<i>EXCSAT</i> (on page 363)
	<i>EXCSQLIMM</i> (on page 371)
	<i>EXCSQLSET</i> (on page 377)
	<i>EXCSQLSTT</i> (on page 381)
	<i>INHERITANCE</i> (on page 437)
	<i>OOPOVR</i> (on page 547)
	<i>OPNQR</i> Y (on page 555)
	<i>PRPSQLSTT</i> (on page 636)
	<i>RDBCMM</i> (on page 728)
	<i>RDBRLLBCK</i> (on page 745)
	<i>REBIND</i> (on page 753)
	<i>RQSDSS</i> (on page 774)
	<i>SECCHK</i> (on page 800)
	<i>SNDPKT</i> (on page 829)
	<i>SUBSETS</i> (on page 902)
	<i>SYNCCTL</i> (on page 915)
	<i>SYNCRSY</i> (on page 982)
	<i>TYPDEFOVR</i> (on page 1030)



**NAME**

CONCEPTS — Concepts of the DDM Architecture

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

Concepts of the DDM architecture (CONCEPTS) are defined by the following terms and their references:

**ABBREVIATIONS** DDM Dictionary Abbreviations (see *ABBREVIATIONS* (on page 30))

**CMNLYR** Communications Layers (see *CMNLYR* (on page 194))

**DDMOBJ** DDM Objectives (see *DDMOBJ* (on page 262))

**DSS** Data Stream Structures (see *DSS* (on page 308))

**EXTENSIONS** Product Extensions to the DDM Architecture (see *EXTENSIONS* (on page 400))

**INHERITANCE** Class Inheritance (see *INHERITANCE* (on page 437))

**LVLCMP** Level Compatibility (see *LVLCMP* (on page 486))

**OOPOVR** Overview of Object-Oriented Programming (see *OOPOVR* (on page 547))

**PRCOVR** DDM Processing Overview (see *PRCOVR* (on page 628))

**STRLYR** Structural Layers of the DDM Architecture (see *STRLYR* (on page 890))

**SUBSETS** Architecture Subsets (see *SUBSETS* (on page 902))

**TASK** Task (see *TASK* (on page 991))

**SEE ALSO**

**Semantic** *DDM* (on page 260)

**NAME**

CONSTANT — Constant Value

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD  
**Codepoint** X'0050'  
**Length** \*  
**Class** CLASS  
**Sprcls** HELP - Help Text

Constant Value (CONSTANT) is a value that cannot be changed.

The instances of class CONSTANT specify a single unchanging value that the title and text variables of the instance describe. The single value is specified as a literal of the required data class.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'0050'	
<b>status</b>	INSTANCE_OF TITLE	STSLST - Term Status Collection term status
<b>title</b>	INSTANCE_OF	TITLE - Title
<b>semantic</b>	INSTANCE_OF MAXLEN REPEATABLE TITLE	TEXT - Text Character String 80 description
<b>value</b>	NIL	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**Semantic** ACCDMG (on page 41)  
 DCESEC (on page 252)  
 DDMID (on page 261)  
 ERROR (on page 350)  
 EUSRIDDTA (on page 355)  
 EUSRNPWDDTA (on page 359)  
 EUSRPWDDTA (on page 361)  
 FALSE (on page 403)  
 INFO (on page 436)  
 OUTOVRANY (on page 574)  
 OUTOVRFRS (on page 575)  
 OWNER (on page 577)  
 PLGIN (on page 609)  
 PRMDMG (on page 632)  
 QRYBLKEXA (on page 674)  
 QRYBLKFLX (on page 676)  
 QRYSCRCUR (on page 706)

*QRYSCRFST* (on page 707)  
*QRYSCRLST* (on page 708)  
*QRYSCRNXT* (on page 709)  
*QRYSCRPRI* (on page 712)  
*REQUESTER* (on page 760)  
*RTNEXTALL* (on page 791)  
*RTNEXTROW* (on page 793)  
*SESDMG* (on page 825)  
*SEVERE* (on page 826)  
*TRUE* (on page 1024)  
*USRENCPWD* (on page 1043)  
*USRIDNWPWD* (on page 1045)  
*USRIDONL* (on page 1046)  
*USRIDPWD* (on page 1047)  
*USRSBSPWD* (on page 1049)  
*WARNING* (on page 1060)

NAME

CRRTKN — Correlation Token

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD

**Codepoint** X'2135'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Correlation Token (CRRTKN) String specifies a token that is conveyed between source and target servers for correlating the processing between the servers. For more information, see the *SNA LU 6.2 Reference: Peer Protocols* (SC31-6808, IBM) and the DRDA Reference.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'2135'	
value	INSTANCE_OF	BYTSTRDR - Byte String
	MINLEN	0
	MAXLEN	255
	REQUIRED	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

SEE ALSO

**insvar** ACCRDB (on page 42)  
ACCRDBRM (on page 48)

**NAME**

CSTBITS — Character Subtype Bits

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2433'**Length** \***Class** codpnt

Character Subtype Bits (CSTBITS) specifies that the target relational database (RDB) uses the FOR BITS DATA SQL character subtype for all new character columns for which an explicit subtype is not specified.

**SEE ALSO****insvar** *PKGDFTCST* (on page 590)

## NAME

CSTMBCS — Character Subtype MBCS

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD

**Codepoint** X'2435'

**Length** \*

**Class** codpnt

Character Subtype MBCS (CSTMBCS) specifies that the target relational database (RDB) uses the FOR MIXED DATA SQL character subtype for all new character columns for which an explicit subtype is not specified.

## SEE ALSO

**insvar** *PKGDFTCST* (on page 590)

**NAME**

CSTSBCS — Character Subtype SBCS

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2434'**Length** \***Class** codpnt

Character Subtype SBCS (CSTSBCS) specifies that the target relational database (RDB) uses the FOR SBCS DATA SQL character subtype for all new character columns an explicit subtype is not specified.

**SEE ALSO****insvar** *PKGDFTCST* (on page 590)

**NAME**

CSTSYSDFT — Character Subtype System Default

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2432'

**Length** \*

**Class** codpnt

Character Subtype System Default (CSTSYSDFT) specifies that the target relational database (RDB) uses the target system-defined default for all new character columns for which an explicit subtype is not specified.

**SEE ALSO**

**insvar** *PKGDFTCST* (on page 590)



**NAME**

DATA — Encoded Information

**DESCRIPTION (Semantic)****Dictionary** QDDPRMD**Codepoint** X'003C'**Length** \***Class** CLASS**Sprcls** NIL

Encoded Information (DATA) serves as the superclass for all means of encoding information for the DDM architecture. It is one of many ways to encode or represent information in a computer system.

Data stream structures (DSSs) encode all forms of DDM information for transmission.

DATA is only used as an abstract class representing the concept of information encoding.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>NIL</b>
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO**

**Semantic** *APPCMNI* (on page 72)  
*APPCMNT* (on page 76)  
*APPSRCCD* (on page 79)  
*APPTRGER* (on page 95)  
*CLASS* (on page 158)  
*DSS* (on page 308)  
*FIELD* (on page 415)  
*HELP* (on page 425)  
*INHERITANCE* (on page 437)  
*LMTBLKPRC* (on page 475)  
*MGRLVL* (on page 506)  
*OBJECT* (on page 540)  
*RQSCRR* (on page 772)  
*STRLYR* (on page 890)  
*SYNCMNBK* (on page 924)  
*SYNCMNCM* (on page 926)  
*SYNCMNFL* (on page 928)  
*SYNCMNI* (on page 931)  
*SYNCMNT* (on page 935)

**NAME**

DCESEC — Distributed Computing Environment Security

**DESCRIPTION (Semantic)****Dictionary** QDDPRMD**Length** \***Class** CONSTANT

The Distributed Computing Environment Security (DCESEC) specifies a security mechanism combination for the ACCSEC command.

See the following terms for the specific meaning and function of the security mechanism:

DCESECOVR DCE Security Overview (see *DCESECOVR* (on page 253))DCESECTKN DCE Security Token (see *DCESECTKN* (on page 257))OSFDCE OSF DCE Security Mechanism (see *OSFDCE* (on page 570))

---

 value 1
**SEE ALSO****cmddta** *SECCHK* (on page 800)**insvar** *SECCHK* (on page 800)  
*SECMEC* (on page 811)**Semantic** *DCESECOVR* (on page 253)  
*SECCHK* (on page 800)  
*SECMEC* (on page 811)

## NAME

DCESECOVR — DCE Security Overview

## DESCRIPTION (Semantic)

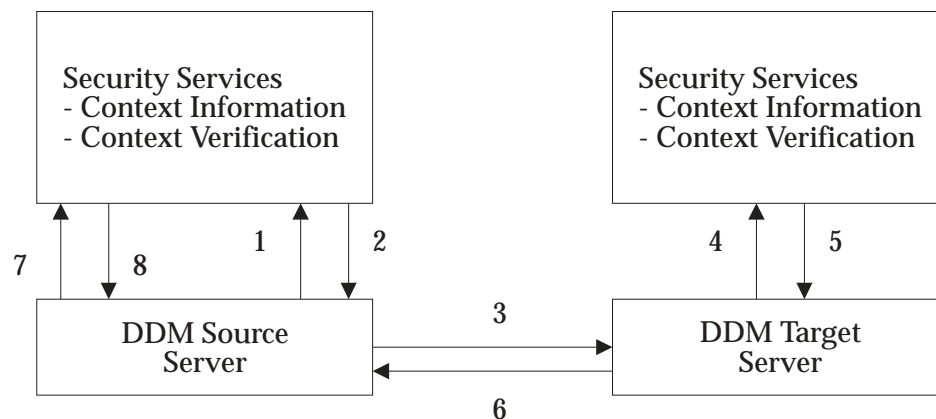
**Dictionary** QDDTTRD**Length** \***Class** HELP

The DCE Security Overview (DCESECOVR) provides an overview of the DDM flows while using the Distributed Computing Environment (DCE) security mechanism.

Figure 3-21 (on page 254) indicates the DDM commands and replies that flow in the normal process of establishing a connection while using the DCE security mechanism for identification and authentication. Figure 3-20 shows the exchange of messages to obtain and use the security context information within the security token.

See the following references for more information on the GSS-API used by DCE:

- *OSF DCE SIG Request For Comments 5.2, GSS-API Extensions for DCE*
- *DCE 1.1: Authentication and Security Services*
- *IETF Request For Comments 1508, Generic Security Service Application Program Interface*
- *IETF Request For Comments 1510, The Kerberos Network Authentication Service (V5)*



**Figure 3-20** DCE-Based Security Flows Using GSS-API

The following is a brief description of the DCE security flows.

1. The DDM source server calls the security services to obtain security context information for accessing the DDM target server. The figure indicates a single flow, but in actuality, there may be several flows. Acting on behalf of the end user of the application, the source DDM server calls the security services using `gss_init_sec_context()` to obtain the security context information for accessing the target DDM server.
2. The security service returns the security context information. Assuming the `gss_init_sec_context()` call is successful, the `major_status` code is `GSS_S_CONTINUE_NEEDED`. The source security service is waiting for more information.
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 the results to the target server, which includes security context information so that the source server can verify the target server. Assuming the *gss\_accept\_sec\_context()* call is successful, the mutual authentication information is returned to the target DDM server.
6. The target server returns the results to the source server.
7. The source server calls the security services to verify the security context information received from the target server. The source DDM server calls the source security services using the *gss\_init\_sec\_context()* passing the security context information received from the target DDM server.
8. The security service returns the results to the source server. Assuming the call *gss\_init\_sec\_context()* is successful, the major\_status code is *GSS\_S\_COMPLETE* signifying the security context information is valid and correct.

See Figure 3-21 for flows to establish a connection using the DCE-based security mechanism.

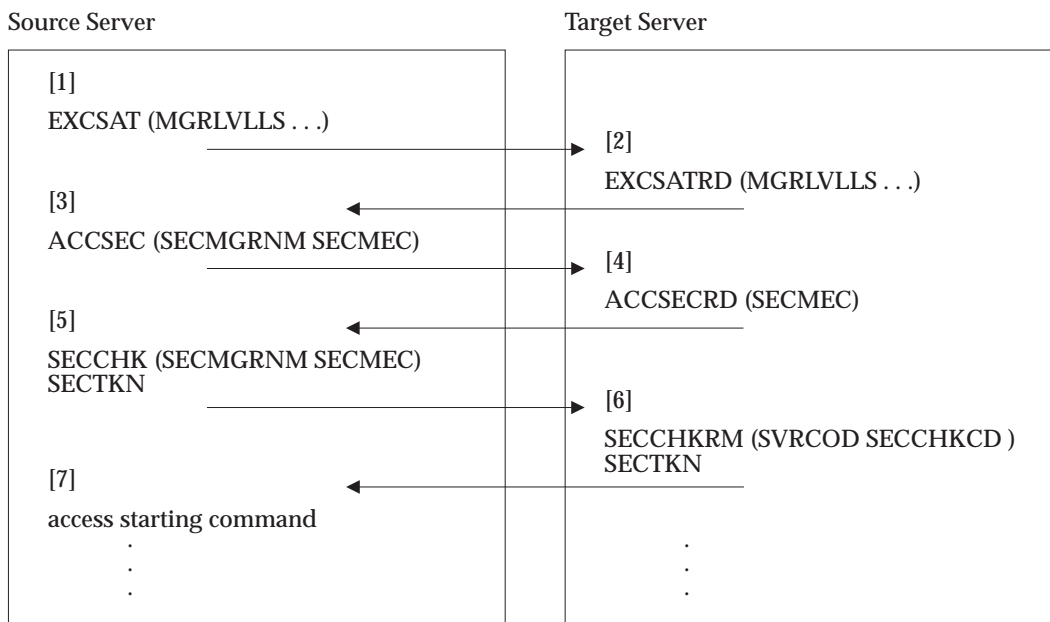


Figure 3-21 DDM DCE Security Flows

The following is a brief description of some of the parameters for the DDM commands. See the appropriate terms a detailed description of the parameters.

1. The source server identifies the level of security it would like to operate at by identifying the SECMGR at manager Level 5 on the EXCSAT command.

```

EXCSAT (
    MGRLVLLS (
        MGRLVL (SECMGR, 5)
        .
        .
    )
)
    
```

.))

2. The target server receives the EXCSAT command and builds a reply data object with the SECMGR at manager Level 5 indicating it can operate at that security level. The target server sends the EXCSATRD reply data to the source server.

```
EXCSATRD (
    MGRLVLLS (
        MGRLVL (SECMGR, 5)
        .
        .
    .))
```

3. The source server receives the EXCSATRD reply data. The type of identification and authentication mechanism is negotiated through the ACCSEC command.

The SECMEC parameter indicates the security mechanism to use. This example assumes the DCESEC security mechanism is specified.

4. The target server receives the ACCSEC command. It supports the security mechanism identified in the SECMEC parameter and returns the value in the ACCSECRD.

If the target server does not support or accept the security mechanism specified in the SECMEC parameter on ACCSEC command, the target server returns the security mechanism combination values that it does support in the SECMEC parameter in the ACCSECRD object.

5. The source server receives the ACCSECRD object and generates the security token containing the security context information required for security processing. The actual process to generate the security context information is not specified by DDM. The source server may either generate the security context information, or it may call a security resource manager to generate the security context information.

The source server passes the security context information in a SECCHK command with a SECTKN object. For information about the DCE security context information, see *DCESECTKN* (on page 257).

6. The target 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 target 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 target server must generate authentication reply security context information to return to the source server.

The target server generates a SECCHKRM and SECTKN to return to the source server. The SECCHKCD parameter identifies the status of the security processing. The SECTKN carries the reply authentication information. A failure to authenticate the end-user or to successfully pass the security processing results in the SVRCOD parameter value set to greater than WARNING.

7. The source server receives the SECCHKRM and SECTKN. Assuming authentication at the target server is successful, then the source server verifies the security context information received in the SECTKN.

Assuming security processing is successful, the source server sends any data access starting command to the target server.

SECCHKCD values that indicate a failure with processing the security information (for example, bad context information, expired context information) require the security be retried or terminate the network connection.

If security processing fails, the source server might attempt recovery before returning to the application. For example, if the context information has expired, the source server could request new security context information to send to the target server. If the error condition is not recoverable, the source server returns to the application with an error indicating a security verification failure.

Figure 3-21 (on page 254) describes the synchronization of security information for a source server and target server pair. If a source server is connecting to multiple target servers, the security information shared between each source server and target server pair is independent from each of the other pairs.

**SEE ALSO**

<b>insvar</b>	<i>SECCHK</i> (on page 800)
<b>Semantic</b>	<i>DCESEC</i> (on page 252)
	<i>DCESECTKN</i> (on page 257)
	<i>SECCHK</i> (on page 800)
	<i>SECMGR</i> (on page 814)
	<i>SECOVR</i> (on page 819)

**NAME**

DCESECTKN — DCE Security Token

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

The DCE Security Token (DCESECTKN) is information provided and used by the DCE security mechanism. See *DCESECOVR* (on page 253) for information on the DDM flows that carry the DCESECTKN. The DCESECTKN is carried in a SECTKN object as command and reply data to the SECCHK command.

The security token contains DCE security context information to identify and authenticate a user to the target server or to authenticate the target server to the source server.

DDM architecture treats the DCESECTKN as a byte string. The SECMGR is aware of and has knowledge of the contents of the DCESECTKN. Sometimes the DCESECTKN may contain the DCE security ticket and other times may contain the DCE security authentication information. DDM does not know or care about the specific contents of this field. If the reader needs to know the actual contents, see the documentation in the overview term.

**SEE ALSO**

**Semantic**        *DCESEC* (on page 252)  
                  *DCESECOVR* (on page 253)

## NAME

DCTIND — Dictionary Index

## DESCRIPTION (Semantic)

**Dictionary** QDDBASD**Codepoint** X'1450'**Length** \***Class** CLASS**Sprcls** INDEX - Index

Dictionary Index (DCTIND) Index maps external names and codepoints to the ordinal numbers of selected dictionaries. Not all dictionary objects have external names or codepoints.

The term **DICTIONARY** provides a model for an implementation of a simple linear index that can be sequentially searched to find objects with a matching name or codepoint. Local data managers can use other structures such as binary radix trees instead of this model to improve performance.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
length	*
class	X'1450'
entry	INSTANCE_OF DCTINDEN - Dictionary Index Entry REPEATABLE
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

## SEE ALSO

**insvar** *DICTIONARY* (on page 286)**Semantic** *DICTIONARY* (on page 286)



**NAME**

DCTINDEN — Dictionary Index Entry

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1451'

**Length** \*

**Class** CLASS

**Sprcls** ASSOCIATION - Name with Value Association

Dictionary Index Entry (DCTINDEN) is an index entry that associates an object's external name or codepoint with the ordinal number where the object is stored in the dictionary.

<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'1451'	
key	ENUCLS	NAME - Name
	ENUCLS	CODPNT - Codepoint
	REQUIRED	
value	INSTANCE_OF	BIN - Binary Integer Number
	LENGTH	32
	REQUIRED	
clscmd	NIL	
inscmd	NIL	

**SEE ALSO**

**insvar** *DCTIND* (on page 258)

**Semantic** *DICTIONARY* (on page 286)

**NAME**

DDM — Distributed Data Management Architecture

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

Distributed Data Management (DDM) Architecture concepts, structures, and protocols are defined by the following terms:

**CONCEPTS** Concepts of the DDM Architecture (see *CONCEPTS* (on page 243))**DTAOVR** Data Layer Overview (see *DTAOVR* (on page 321))**MGROVR** Manager Layer Overview (see *MGROVR* (on page 514))**OBJOVR** Object Layer Overview (see *OBJOVR* (on page 544))**SRVOVR** Server Layer Overview (see *SRVOVR* (on page 881))**SEE ALSO**

None.

**NAME**

DDMID — DDM Identifier

**DESCRIPTION (Semantic)****Dictionary** QDDPRMD**Length** \***Class** CONSTANT

DDM Identifier (DDMID) is the SNA registered General Data Stream (GDS) identifier for all DDM data stream structures.

---

value	X'D0'
-------	-------

**SEE ALSO**

<b>insvar</b>	<i>OBJDSS</i> (on page 536)
	<i>RPYDSS</i> (on page 766)
	<i>RQSDSS</i> (on page 774)

<b>Semantic</b>	<i>DSS</i> (on page 308)
-----------------	--------------------------

## NAME

DDMOBJ — DDM Objectives

## DESCRIPTION (Semantic)

**Dictionary** QDDTTRD**Length** \***Class** HELP

DDM Objectives (DDMOBJ) describe the data communication networks and objectives of DDM architecture.

Data communication networks allow data and users to be geographically or organizationally distributed. These networks can combine:

1. Multiple systems of the same kind
2. Multiple systems of different kinds
3. Multiple kinds of operating systems

The general problems in constructing a network are:

1. A variety of interfaces manage and access file data.
2. Data and users are distributed throughout networks of heterogeneous systems, workstations, and file servers.
3. Each system product and access method supports its own unique set of functions.
4. As the number of systems in the network increases, interconnection becomes more expensive.
5. As the number of systems in the network increases, adding or changing functions becomes more difficult.

The objectives of the DDM architecture are:

- Data interchange among different kinds of currently existing systems (heterogeneous)
- Efficient data interchange among homogeneous systems
- Standardized data management facilities for new systems
- Evolution of new forms of data management

These objectives seem to conflict but only because no conceptual framework existed that:

- Encompasses existing data management systems without modification
- Directs the gradual migration of existing data management systems toward a common model of data management
- Focuses the design of new systems on common models
- Is open to the development of new data management models

One solution is to pick one system's interfaces and translate the interfaces of all other systems to them. Another solution is to create a new set of common interfaces. The first solution is not plausible because the enhancements and quirks already built into existing systems are unacceptable to all other systems. Therefore, a common interface is needed. DDM provides an open-ended set of abstract models that serve as the basis for data connectivity.

**SEE ALSO**

**Semantic**      *CONCEPTS* (on page 243)

**NAME**

DECDELCMA — Decimal Delimiter is Comma

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'243D'**Length** \***Class** codpnt

Decimal Delimiter is Comma (DECDELCMA) specifies that the decimal delimiter in the SQL statements is the comma (with the Graphic Character Global Identifier (GCGID) SP08) character. For more information on GCGID, see *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO****insvar** *STTDECDEL* (on page 897)

**NAME**

DECDELPRD — Decimal Delimiter is Period

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'243C'

**Length** \*

**Class** codpnt

Decimal Delimiter is Period (DECDELPRD) specifies that the decimal delimiter in the SQL statements is the period (with the Graphic Character Global Identifier (GCGID) SP11) character. For more information on GCGID, see *Character Data Representation Architecture Registry* (SC09-1931, IBM).

**SEE ALSO**

**insvar** *STTDECDEL* (on page 897)

**NAME**

DECPRC — Decimal Precision

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2106'

**Length** \*

**Class** CLASS

**Sprcls** BIN - Binary Integer Number

Decimal Precision (DECPRC) specifies the decimal precision used during decimal arithmetic processing at the target database. The decimal arithmetic rules that apply depend on the precision in effect.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'2106'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	16
	ENUVAL	15
	ENUVAL	31
	ENUVAL	63
	NOTE	The default value means that the target RDB selects the target product-specific decimal precision for arithmetic computations.
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *BGNBND* (on page 110)



## NAME

DEFINITION — Definition

## DESCRIPTION (Semantic)

**Dictionary** QDDPRMD**Codepoint** X'0048'**Length** \***Class** CLASS**Sprcls** ASSOCIATION - Name with Value Association

Definition (DEFINITION) associates a name with an attribute list. Definitions specify the characteristics of variables, values, and other aspects of objects.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'0048'	
key	ELMCLS REQUIRED	NAME - Name
value	SPRCLS REQUIRED	ATTLIST - Attribute List
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

## SEE ALSO

**insvar** *DEFLST* (on page 268)**Semantic** *DEFLST* (on page 268)

NAME

DEFLST — Definition List

DESCRIPTION (Semantic)

**Dictionary** QDDPRMD

**Codepoint** X'0047'

**Length** \*

**Class** CLASS

**Sprcls** INDEX - Index

Definition List (DEFLST) Index specifies a list of definitions.

Definition lists specify the definitions of a set of variables, commands, or other aspects of objects in descriptions of objects.

The overall structure of a definition list, as it would be mapped into a dictionary with its component sub-objects, is illustrated in Figure 3-22.

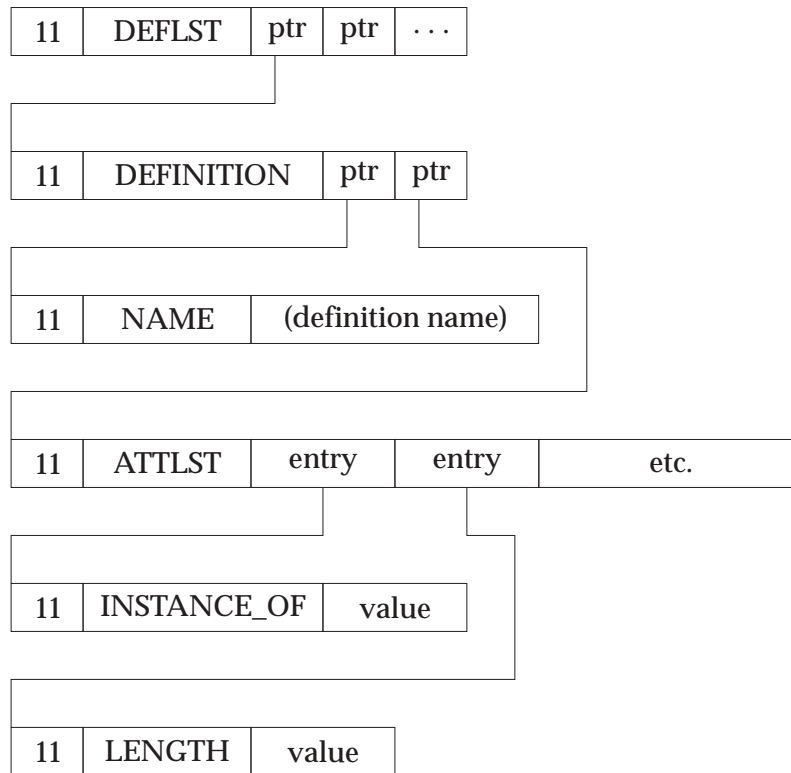


Figure 3-22 Structure of a Definition List

---

clsvar	NIL
--------	-----

---

<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'0047'	
entry	INSTANCE_OF	DEFINITION - Definition
	REPEATABLE	
	REQUIRED	
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

<b>clscmd</b>	<i>CLASS</i> (on page 158)
<b>clsvar</b>	<i>CLASS</i> (on page 158)
<b>cmddda</b>	<i>COMMAND</i> (on page 240)
<b>cmdrpy</b>	<i>COMMAND</i> (on page 240)
<b>inscmd</b>	<i>CLASS</i> (on page 158)
<b>insvar</b>	<i>CLASS</i> (on page 158)
<b>mgrdepls</b>	<i>MANAGER</i> (on page 491)
<b>rpydta</b>	<i>COMMAND</i> (on page 240)
<b>vldattls</b>	<i>FIELD</i> (on page 415) <i>MANAGER</i> (on page 491)
<b>Semantic</b>	<i>CLASS</i> (on page 158) <i>INHERITED</i> (on page 442)

NAME

DEPERRCD — Manager Dependency Error Code

DESCRIPTION (Semantic)

**Dictionary** QDDBASD

**Codepoint** X'119B'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

The Manager Dependency Error Code (DEPERRCD) String specifies the manager dependency that was not satisfied.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	5	
class	X'119B'	
value	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	LENGTH	2
	ENUVAL	X'01'
	NOTE	This value means the communications manager has been allocated with parameters other than those the SQLAM manager requires.
	ENUVAL	X'02'
	NOTE	This value means the SUPERVISOR manager has not received on the EXCSAT command the information that the SQLAM manager needs.
	ENUVAL	X'03'
	MINLVL	5
	NOTE	This value means the SECMGR has not received a verified end user name.
	ENUVAL	X'04'
	MINLVL	5
	NOTE	This value means the RSYNCMGR manager requested does not support resync, but the conversation was established using a network address that supports only resync requests.
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar**

*MGRDEPRM* (on page 505)

**NAME**

DFTDATFMT — Default Date Format

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2400'**Length** \***Class** CODPNT

The Default Date Format (DFTDATFMT) specifies the default format for dates in the Structured Query Language (SQL) statements. The default date format is determined based on the locale used by the application at the source server and must be identical to one of the other predefined date formats (including LOCDATFMT) in DDM.

The means for conveying the relevant locale information of the application from the source server to the target server is implementation-specific and is not architected by DDM. For instance, the source server may achieve this objective by flowing a non-local SQL SET LOCALE statement to the target server.

**SEE ALSO****insvar** *STTDATFMT* (on page 894)**Semantic** *CODPNT* (on page 233)  
*LOCDATFMT* (on page 482)  
*QDDRDBD* (on page 657)

**NAME**

DFTPKG — Package Default

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'241E'**Length** \***Class** codpnt

Package Default (DFTPKG) is a parameter that helps define the SQL statement delimiters.

If specified as a value of the STTDECDEL parameter on the ACCRDB command, the decimal delimiter for the subsequent dynamic SQL statements is taken from the package through which the dynamic SQL statements are executed.

If specified as a value of the STTSTRDEL parameter on the ACCRDB command, the characters used for delimiting character strings and delimited SQL identifiers for the subsequent dynamic SQL statements are taken from the package through which the dynamic SQL statements are executed.

This parameter value is not valid if specified on the BGNBND command.

**SEE ALSO**

**insvar**            *STTDECDEL* (on page 897)  
                  *STTSTRDEL* (on page 899)

## NAME

DFTRDBCOL — Default RDB Collection Identifier

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD**Codepoint** X'2128'**Length** \***Class** CLASS**Sprcls** NAME - Name

Default RDB Collection Identifier (DFTRDBCOL) specifies the relational database (RDB) collection identifier that the target RDB uses to complete the RDB object names if necessary for the SQL statements bound into the RDB package.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'2128'	
name	INSTANCE_OF	CHRSTRDR - Character String Data Representation
	MAXLEN	255
	DFTVAL	''
	NOTE	The default value means that the target RDB selects a default RDB collection ID value used for RDB object name completion.
	OPTIONAL	
	MINLVL	7
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

## SEE ALSO

**insvar** *BGNBND* (on page 110)  
*CHRSTRDR* (on page 156)  
*REBIND* (on page 753)



**NAME**

DFTTIMFMT — Default Time Format

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2401'**Length** \***Class** CODPNT

The Default Time Format (DFTTIMFMT) specifies the default format for times in the Structured Query Language (SQL) statements. The default time format is determined based on the locale used by the application at the source server and must be identical to one of the other predefined time formats (including LOCTIMFMT) in DDM.

The means for conveying the relevant locale information of the application from the source server to the target server is implementation-specific and is not architected by DDM. For instance, the source server may achieve this objective by flowing a non-local SQL SET LOCALE statement to the target server.

**SEE ALSO****insvar** *STTIMFMT* (on page 900)**Semantic** *CODPNT* (on page 233)  
*LOCTIMFMT* (on page 483)  
*QDDRDBD* (on page 657)

NAME

DFTVAL — Default Value Attribute

DESCRIPTION (Semantic)

**Dictionary** QDDPRMD  
**Codepoint** X'0011'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

Default Value Attribute (DFTVAL) Scalar Object specifies that this attribute value is used as the default value for an optional parameter or value that the requester did not specify.

The attribute value must have attributes compatible with those of the term being described.

When the default value is given as a double apostrophe ("), a note attribute always follows and explains what value the server should use for the parameter.

When a default value attribute is specified for a parameter, the object (length, codepoint, and value) need not be sent.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'0011'	
value	SPRCLS NOTE	FIELD - A Discrete Unit of Data The attributes for this variable vary according to what is being defined. Also, when BITDR, BITSTRDR, HEXDR, or HEXSTRDR values are specified, they are represented as being left-justified in the fewest number of whole bytes possible. For example, 123 would be X'1230'.
	SPCVAL NOTE	” The receiving server determines the default value.
	REQUIRED	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

SEE ALSO

**cmdtda** BGNBND (on page 110)  
 BNDSQLSTT (on page 136)  
 DSCRDBTBL (on page 300)  
 EXCSQLIMM (on page 371)  
 EXCSQLSTT (on page 381)  
 PRPSQLSTT (on page 636)

**insvar** ACCRDBRM (on page 48)  
 ATMIND (on page 104)  
 BGNBND (on page 110)  
 BNDCHKEXS (on page 125)

*BNDCRTCTL* (on page 127)  
*BNDEXPOPT* (on page 129)  
*CNTQRY* (on page 222)  
*DECPRC* (on page 266)  
*DFTRDBCOL* (on page 274)  
*DGRIOPRL* (on page 279)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSTT* (on page 381)  
*FDODSCOFF* (on page 407)  
*FDODTAOFF* (on page 409)  
*FORGET* (on page 423)  
*MAXBLKEXT* (on page 494)  
*MAXRSLCNT* (on page 497)  
*MAXSCTNBR* (on page 498)  
*NBRROW* (on page 530)  
*OPNQRY* (on page 555)  
*OUTOVROPT* (on page 576)  
*PKGATHRUL* (on page 581)  
*PKGDFTCC* (on page 589)  
*PKGDFTCST* (on page 590)  
*PKGOWNID* (on page 600)  
*PKGRPLVRS* (on page 605)  
*PRCNAM* (on page 627)  
*QRYROWNBR* (on page 694)  
*RDBALWUPD* (on page 726)  
*RDBCMTOK* (on page 731)  
*RDBRLSOPT* (on page 750)  
*REBIND* (on page 753)  
*RLSCONV* (on page 765)  
*RSLSETFLG* (on page 783)  
*RTNEXTDTA* (on page 792)  
*RTNSQLDA* (on page 795)  
*SQLCSRHLD* (on page 858)  
*STTDATFMT* (on page 894)  
*STTDECDEL* (on page 897)  
*STTSTRDEL* (on page 899)  
*STTTIMFMT* (on page 900)  
*SYNCCTL* (on page 915)  
*TYPESQLDA* (on page 1034)  
*VRSNAM* (on page 1059)

**rpydta**

*BNDSQLSTT* (on page 136)  
*CLSQRY* (on page 165)  
*CNTQRY* (on page 222)  
*DRPPKG* (on page 293)  
*DSCRDBTBL* (on page 300)  
*DSCSQLSTT* (on page 304)  
*ENDBND* (on page 336)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSTT* (on page 381)  
*OPNQRY* (on page 555)  
*PRPSQLSTT* (on page 636)  
*RDBCMM* (on page 728)

	<i>RDBRLLBCK</i> (on page 745)
	<i>REBIND</i> (on page 753)
<b>Semantic</b>	<i>LVLCMP</i> (on page 486)

**NAME**

DGRIOPRL — Degree of I/O Parallelism

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'212F'  
**Length** \*  
**Class** CLASS  
**Sprcls** BIN - Binary Integer Number

Degree of I/O Parallelism (DGRIOPRL) Binary Integer Number allows an application server to determine if I/O parallel processing is in effect for static SQL queries bound in a package.

clsvar	NIL	
insvar	CLASS INSTANCE VARIABLES	
length	6	
class	X'212F'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	16
	MINVAL	1
	NOTE	A value of one indicates that no I/O parallel processing is done for static SQL queries bound in a package (degree of one).
	SPCVAL	-1
	NOTE	A value of minus one indicates that the RDB uses I/O parallel processing for static SQL queries bound in a package. The RDB uses whatever degree of parallel processing it determines is appropriate. Values of two through 32,767 restrict the degree of I/O parallel processing for static SQL queries bound in a package.
	DFTVAL	1
	NOTE	The default value of one indicates that no I/O parallel processing is done for static SQL queries (degree of one).

**SEE ALSO**

**insvar** *BGNBND* (on page 110)  
*REBIND* (on page 753)

## NAME

DHENC — Diffie-Hellman Encryption Security Mechanism

## DESCRIPTION (Semantic)

**Dictionary** QDDTTRD

**Length** \*

**Class** HELP

The Diffie-Hellman Encryption Security Mechanism (DHENC) specifies the method of encrypting and decrypting data. See *USRSECOVR* (on page 1050) and *EDTASECOVR* (on page 323) for more information about this mechanism.

A shared private key is generated by exchanging connection keys in the SECTKN instance variable on the ACCSEC command and the ACCSECRD reply data, using the standard Diffie-Hellman key distribution algorithm. The encryption algorithm and the encryption key length are exchanged on the ACCSEC command and the ACCSECRD reply data by passing the encryption algorithm in the ENCALG instance variable, and encryption key length string in the ENCKEYLEN instance variable. The default encryption algorithm is DES and the default encryption key length is 56 bits. The encryption seed is generated from the Diffie-Hellman shared private key, as specified by the combination of encryption algorithm and encryption key length. The application requester encrypts the data using the encryption token and the encryption seed generated from the Diffie-Hellman shared private key. The encryption token used depends on the type of security mechanism being used:

- **USRENCPWD:** The user ID is used as the encryption token. The user ID is passed in the USRID instance variable and the encrypted password is passed in the SECTKN instance variable on the SECCHK command. The USRID is zero-padded to 8 bytes if less than 8 bytes or truncated to 8 bytes if greater than 8 bytes. The application server decrypts the password using the shared private key and the user ID. The plain text information is then passed to the local security server.
- **EUSRIDPWD:** The combination of encryption algorithm and encryption key length is used to determine the encryption token. The default encryption token is the middle 8 bytes of the server's connection key. The user ID and password are encrypted and passed in two SECTKN instance variables on the SECCHK command, user ID SECTKN first, password SECTKN second. The application server decrypts the user ID and password using the encryption seed generated from the Diffie-Hellman shared private key and the encryption token, as specified by the combination of encryption algorithm and encryption key length exchanged. The plain text information is then passed to the local security server.
- **EUSRIDNWPWD:** The combination of encryption algorithm and encryption key length is used to determine the encryption token. The default encryption token is the middle 8 bytes of the server's connection key. The user ID, password, and new password are encrypted and passed in three SECTKN instance variables on the SECCHK command, user ID SECTKN first, password SECTKN second, new password third. The application server decrypts the user ID, password, and new password using the encryption seed generated from the Diffie-Hellman shared private key and the encryption token, as specified by the combination of encryption algorithm and encryption key length exchanged. The plain text information is then passed to the local security server.
- **EUSRIDDTA:** The combination of encryption algorithm and encryption key length is used to determine the encryption token. The default encryption token is the middle 8 bytes of the server's connection key. The user ID is encrypted and passed in the SECTKN instance variable on the SECCHK command. The application server decrypts the user ID using the

encryption seed generated from the Diffie-Hellman shared private key and the encryption token, as specified by the combination of encryption algorithm and encryption key length exchanged. The plain text information is then passed to the local security server. If the request data stream contains security-sensitive objects, then the DSS carrier containing the security-sensitive objects is encrypted. The application server decrypts the encrypted objects. If the reply data stream contains security-sensitive objects, then the DSS carrier containing the security-sensitive objects is encrypted. The application requester decrypts the encrypted objects.

- **EUSRPWDDTA:** The combination of encryption algorithm and encryption key length is used to determine the encryption token. The default encryption token is the middle 8 bytes of the server's connection key. The user ID and password are encrypted and passed in two SECTKN instance variables on the SECCHK command, user ID SECTKN first, password SECTKN second. The application server decrypts the user ID and password using the encryption seed generated from the Diffie-Hellman shared private key and the encryption token, as specified by the combination of encryption algorithm and encryption key length exchanged. The plain text information is then passed to the local security server. If the request data stream contains security-sensitive objects, then the DSS carrier containing the security-sensitive objects is encrypted. The application server decrypts the encrypted objects. If the reply data stream contains security-sensitive objects, then the DSS carrier containing the security-sensitive objects is encrypted. The application requester decrypts the encrypted objects.
- **EUSRNPWDDTA:** The combination of encryption algorithm and encryption key length is used to determine the encryption token. The default encryption token is the middle 8 bytes of the server's connection key. The user ID, password, and new password are encrypted and passed in three SECTKN instance variables on the SECCHK command, user ID SECTKN first, password SECTKN second, new password SECTKN third. The application server decrypts the user ID, password, and new password using the encryption seed generated from the Diffie-Hellman shared private key and the encryption token, as specified by the combination of encryption algorithm and encryption key length exchanged. The plain text information is then passed to the local security server. If the request data stream contains security-sensitive objects, then the DSS carrier containing the security-sensitive objects is encrypted. The application server decrypts the encrypted objects. If the reply data stream contains security-sensitive objects, then the DSS carrier containing the security-sensitive objects is encrypted. The application requester decrypts the encrypted objects.

### Generating the Shared Private Key

The Diffie-Hellman algorithm is used to generate a shared private key between the application requester and application server. This secret key is used as the encryption seed to the DES encryption algorithm. Diffie-Hellman is the first standard public key algorithm ever invented. It gets its security from the difficulty of calculating discrete logarithms in a finite field. Diffie-Hellman requires agreed values  $n$  and  $g$ , such that  $g$  is a primitive of large prime  $n$  and the size of the exponent. These values are fixed. First, the application requester chooses a random large integer  $x$  and generates the value  $X$  where  $X = g^x \text{ mod } n$ .  $X$  is sent in the SECTKN on the ACCSEC command. Second, the application server chooses another random large integer  $y$  and generates the value  $Y$  where  $Y = g^y \text{ mod } n$ .  $Y$  is sent in the SECTKN on the reply to the ACCSEC command. The application requester computes the shared private key,  $k = Y^x \text{ mod } n$ . The application server computes the shared private key,  $k1 = X^y \text{ mod } n$ . The middle 8 bytes of the 32-byte  $k$  are used as the DES encryption key.

Following are DRDA's Diffie-Hellman agreed public values for  $g$  (generator),  $n$  (prime), and the size of the exponent ( $x$  and  $y$ ). These values must be used as-is to generate a shared private key:

```

static unsigned int prime_len = 32;
static unsigned char prime[32] = {
0xc6, 0x21, 0x12, 0xd7, 0x3e, 0xe6, 0x13, 0xf0,
0x94, 0x7a, 0xb3, 0x1f, 0x0f, 0x68, 0x46, 0xa1,
0xbf, 0xf5, 0xb3, 0xa4, 0xca, 0x0d, 0x60, 0xbc,
0x1e, 0x4c, 0x7a, 0x0d, 0x8c, 0x16, 0xb3, 0xe3
};

static unsigned int generator_len = 32;
static unsigned char generator[32] = {
0x46, 0x90, 0xfa, 0x1f, 0x7b, 0x9e, 0x1d, 0x44,
0x42, 0xc8, 0x6c, 0x91, 0x14, 0x60, 0x3f, 0xde,
0xcf, 0x07, 0x1e, 0xdc, 0xec, 0x5f, 0x62, 0x6e,
0x21, 0xe2, 0x56, 0xae, 0xd9, 0xea, 0x34, 0xe4
};

static unsigned int exponent_bits = 255;

```

### DES Encryption

The Data Encryption Standard (DES), known as the Data Encryption Algorithm (DEA) by ANSI and the DEA-1 by ISO, is the worldwide standard for encrypting data. DES is a block cipher; it encrypts data in 64-bit blocks.

DRDA encryption uses DES CBC mode as defined by the FIPS standard (FIPS PUB 81). DES CBC requires an encryption key and an encryption token to encrypt the data. Again, the encryption token to be used depends on the security mechanism being used. If the security mechanism requires the exchange of encryption algorithm and encryption key length and if the encryption algorithm is DES and the encryption key length is 56 bits, then the middle 8 bytes of the server's connection key is used as the encryption token.

If the encryption algorithm is DES and the encryption key length is 56 bits, then the encryption key length is 56 bits. The Diffie-Hellman shared private key is 256 bits. To reduce the number of bits, 64 bits are selected from the Diffie-Hellman shared private key by selecting the middle 8 bytes and parity is added to the lower order bit of each byte producing a 56-bit key with 8 bits of parity.

To decrypt the encrypted data, DES CBC mode requires the encryption token and the encryption key.

The DES CBC mode is a block method of encryption and must operate on 64-bit data blocks. For partial data blocks (blocks of less than 64 bits), DRDA encryption uses the method as defined by the PKCS #5 padding scheme, FIPS standard (FIPS PUB 81). According to the PKCS #5 padding scheme, the padding string should consist of  $8 - (\text{Block-length} \bmod 8)$  bytes all having value  $8 - (\text{Block-length} \bmod 8)$ . In other words, the encrypted data block (EB) satisfies one of the following statements:

```

EB = Data block || 01 -- if block-length mod 8 = 7
EB = Data block || 0202 -- if block-length mod 8 = 6
...
...
EB = Data block || 0808080808080808 -- if block-length mod 8 = 0

```

Hence, the final partial data block is padded with the value as the number of padding bytes needed and a padding indicator denoting the number of padding bytes, including itself is placed in the least significant byte of the input block before encrypting. The cipher text message is marked as being padded. The decrypter reverses the padding process, scans the decrypted



padded block, discards the least significant bits that are all identical, and produces the original plain text.

**SEE ALSO**

**Semantic**      *EDTASECOVR* (on page 323)  
*EUSRIDDTA* (on page 355)  
*EUSRIDPWD* (on page 358)  
*EUSRPWDDTA* (on page 361)

NAME

DIAGLVL — SQL Error Diagnostic Level

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'2160'  
**Length** \*  
**Class** HEXSTRDR  
**Sprcls** STRING - String  
**Clsvar** NIL

The diagnostic level object identifies the amount of diagnostic information requested for an SQL statement that fails or returns warnings. Diagnostic information is returned in an SQLCARD reply object. The source SQLAM can request the default or extended diagnostics. If extended is requested, the target SQLAM should provide a non-null SQLDIAGGRP group with the SQLCARD. The SQLDIAGGRP contains warnings or additional diagnostic information on why the SQL statement failed.

If DIAGLVL0 is specified or the default value is provided, the SQLDIAGGRP is returned as a null group. The DIAGLVL0 may be upgraded by the target server to DIAGLVL1 when executing an EXCSQLSTT command or an OPNQRY command which generates multiple error or warning conditions processing the command. For example, a multi-row SQL request can generate multiple errors and warnings per request.

If DIAGLVL1 is specified, the SQLDIAGGRP should be returned as a non-null group.

If DIAGLVL2 is specified, the SQLDIAGGRP is returned as a non-null group, and both SQLDCMSG message text fields should be returned as null strings.

For more information on the SQLCARD FD:OCA descriptor, refer to the DRDA Reference.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	5	
class	X'2160'	
value	INSTANCE_OF	HEXSTRDR
	LENGTH	2
	ENUVAL	X'F0'
	NOTE	Default Diagnostics.
	NOTE	Any SQLCARD must contain a null SQLDIAGGRP group.
	ENUVAL	X'F1'
	NOTE	Extended Diagnostics with message text.
	NOTE	Any SQLCARD is requested to contain the SQLDIAGGRP group.
	ENUVAL	X'F2'
	NOTE	Extended Diagnostics without message text.

NOTE	SQLCARD generated without any message text.
MINLVL	7
DFTVAL	X'F0'

**SEE ALSO****insvar**

*ACCRDB* (on page 42)  
*HEXSTRDR* (on page 429)  
*SQLCARD* (on page 855)

**Semantic**

*ACCRDB* (on page 42)  
*HEXSTRDR* (on page 429)  
*QDDRDBD* (on page 657)  
*SQLAM* (on page 847)  
*SQLCARD* (on page 855)

**NAME**

DICTIONARY — Dictionary

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1458'

**Length** \*

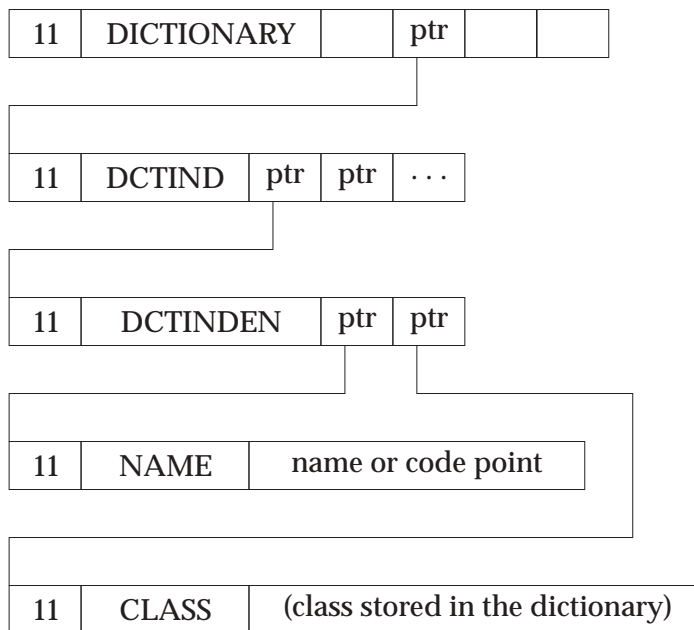
**Class** CLASS

**Sprcls** MANAGER - Resource Manager

Dictionary (DICTIONARY) is a manager of a set of named descriptions of objects. Dictionaries are one of the basic operative components of a DDM file server.

The primary objects stored in a dictionary are instances of classes HELP and CLASS. Each of these objects has an external name and an external codepoint which locate the object. Because these are complex objects (nested collections of many sub-objects), most objects in a dictionary file do not have external names or codepoints.

The entries of a dictionary are of varying lengths and each contains a single complete object. For scalar objects, all of the object's data immediately follows the length and class codepoint of the object. For collection objects, the data following the length and class codepoint of the collection consist of four-byte binary numbers specifying the entry number in the dictionary at which the collection entry is stored. This is illustrated in Figure 3-23.



**Figure 3-23** Dictionary Structure Overview

**Figure Notes**

- Each structure is an object with the name of its class and selected variables specified.
- Arrows denote references to independent objects stored in the dictionary.
- Ellipsis (...) denotes repeated variables.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'1458'	
dctind	INSTANCE_OF	DCTIND - Dictionary Index
objlst	*	
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>mgrlvln</b>	1	
<b>mgrdepls</b>	<b>NIL</b>	
<b>vldattls</b>	<b>VALID ATTRIBUTES</b>	
X'0019'	INSTANCE_OF	HELP - Help Text
X'1452'	INSTANCE_OF	MGRNAM - Manager Name
X'0045'	INSTANCE_OF	TITLE - Title

**SEE ALSO**

<b>insvar</b>	<i>ACCSEC</i> (on page 52) <i>MGRLVL</i> (on page 506)
<b>mgrdepls</b>	<i>AGENT</i> (on page 61)
<b>Semantic</b>	<i>ABNUOWRM</i> (on page 39) <i>ACCDMG</i> (on page 41) <i>ACCRDB</i> (on page 42) <i>ACCRDBRM</i> (on page 48) <i>AGENT</i> (on page 61) <i>DCTIND</i> (on page 258) <i>EXTENSIONS</i> (on page 400) <i>INHERITANCE</i> (on page 437) <i>MANAGER</i> (on page 491) <i>MGROVR</i> (on page 514) <i>OBJOVR</i> (on page 544) <i>QDDBASD</i> (on page 651) <i>QDDPRMD</i> (on page 654) <i>QDDRDBD</i> (on page 657) <i>QDDTTRD</i> (on page 662) <i>RDBOVR</i> (on page 740) <i>SUBSETS</i> (on page 902)

**NAME**

DMYBLKDATFMT — DMY with Blank Separator Date Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2402'

**Length** \*

**Class** CODPNT

The DMY (Day, Month, Year) with Blank Separator Date Format (DMYBLKDATFMT) specifies that dates in the Structured Query Language (SQL) statements are in the following DMY date format:

dd mm yy

where the " " separator is a blank (with the Graphic Character Global Identifier (GCGID) SP01). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar** *STTDATFMT* (on page 894)

**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)

**NAME**

DMYCMADATFMT — DMY with Comma Separator Date Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2403'

**Length** \*

**Class** CODPNT

The DMY (Day, Month, Year) with Comma Separator Date Format (DMYCMADATFMT) specifies that dates in the Structured Query Language (SQL) statements are in the following DMY date format:

dd, mm, yy

where the "," separator is a comma (with the Graphic Character Global Identifier (GCGID) SP08). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar** *STTDATFMT* (on page 894)

**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)

**NAME**

DMYHPNDATFMT — DMY with Hyphen Separator Date Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2404'

**Length** \*

**Class** CODPNT

The DMY (Day, Month, Year) with Hyphen Separator Date Format (DMYHPNDATFMT) specifies that dates in the Structured Query Language (SQL) statements are in the following DMY date format:

dd-mm-yy

where the "-" separator is a hyphen (with the Graphic Character Global Identifier (GCGID) SP10). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar** *STTDATFMT* (on page 894)

**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)



**NAME**

DMYPRDDATFMT — DMY with Period Separator Date Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2406'

**Length** \*

**Class** CODPNT

The DMY (Day, Month, Year) with Period Separator Date Format (DMYPRDDATFMT) specifies that dates in the Structured Query Language (SQL) statements are in the following DMY date format:

dd.mm.yy

where the "." separator is a period (with the Graphic Character Global Identifier (GCGID) SP11). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar** *STTDATFMT* (on page 894)

**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)

**NAME**

DMYSLHDATFMT — DMY with Slash Separator Date Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2409'

**Length** \*

**Class** CODPNT

The DMY (Day, Month, Year) with Slash Separator Date Format (DMYSLHDATFMT) specifies that dates in the Structured Query Language (SQL) statements are in the following DMY date format:

dd/mm/yy

where the "/" separator is a slash (with the Graphic Character Global Identifier (GCGID) SP12). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar** *STTDATFMT* (on page 894)

**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)

**NAME**

DRPPKG — Drop RDB Package

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2007'**Length** \***Class** CLASS**Sprcls** COMMAND - Command

Drop RDB Package (DRPPKG) Command drops a named package from a relational database (RDB). A package cannot be dropped if it is being bound to the RDB.

**Source System Processing**

The source system determines the location of the RDB:

Local            Call the local RDB server.

Remote          Send the DRPPKG command to the remote RDB server.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**DSS Carrier: RQSDSS****Target System Processing**

The *pkgnam* parameter specifies the fully qualified package name of the package to be dropped from the RDB.

The *rdbnam* parameter specifies the name of the RDB that the ACCRDB command accesses. If the *rdbnam* parameter is specified, its value must be the same as the value specified on the ACCRDB command for RDBNAM.

The *vrsnam* parameter specifies the version name of the package to be dropped. The version name selects between packages that have the same package name.

The TYPDEFNAM reply data object specifies the name of the data type to data representation mapping definitions that the target SQLAM uses when sending the SQLCARD object for this command.

The TYPDEFOVR reply data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type to data representation mapping definitions for the SQLCARD object for this command.

Normal completion of the DRPPKG command returns an SQLCARD object. An RDBUPDRM must be returned if the command makes recoverable updates in the RDB, and the server in the current unit of work did not return a prior RDBUPDRM. A recoverable update is an RDB update that writes information to the RDBs recovery log.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**Exceptions**

Exception conditions the RDB detects are reported in an SQLCARD object.

If the bind process is active, then the command is rejected with the PKGBPARM.

If the specified RDB is not currently accessed, then the command is rejected with the RDBNACRM.

**Source System Reply Processing**

Return any reply messages and data objects to the requester.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'2007'	
pkgnam	INSTANCE_OF REQUIRED	PKGNAME - RDB Package Name
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAME - Relational Database Name
vrsnam	INSTANCE_OF OPTIONAL	VRSNAME - Version Name
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	
<b>cmddta</b>	NIL	
<b>rpydta</b>	REPLY OBJECTS	
X'002F'	INSTANCE_OF	TYPDEFNAM - Data Type Definition Name
	ENUVAL	'QTDSQL370'
	ENUVAL	'QTDSQL400'
	ENUVAL	'QTDSQLX86'
	ENUVAL	'QTDSQLASC'
	MINLVL	4
	ENUVAL	'QTDSQLVAX'
	MINLVL	4
	DFTVAL	''

	NOTE	The default means the value received on ACCRDBRM is used.
	OPTIONAL	
X'0035'	INSTANCE_OF DFTVAL	TYPDEFOVR - TYPDEF Overrides ''
	NOTE	The default means the value received on ACCRDBRM is used.
	OPTIONAL	
X'1C03'	INSTANCE_OF OPTIONAL MINLVL	MGRVLVLOVR - Manager Level Overrides  7
X'2408'	INSTANCE_OF REQUIRED	SQLCARD - SQL Communications Area Reply Data
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'220D'	INSTANCE_OF	ABNUOWRM - Abnormal End Unit of Work Condition
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported
X'2209'	INSTANCE_OF	PKGBPARM - RDB Package Binding Process Active
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'2204'	INSTANCE_OF	RDBNACRM - RDB Not Accessed
X'2218'	INSTANCE_OF MINLVL	RDBUPDRM - RDB Update Reply Message 4
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limits Reached
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'125F'	INSTANCE_OF	TRGNSPRM - Target Not Supported
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

**SEE ALSO**

<b>inscmd</b>	<i>SQLAM</i> (on page 847)
<b>Semantic</b>	<i>BGNBND</i> (on page 110) <i>SQLAM</i> (on page 847)

NAME

DSCERRCD — Description Error Code

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'2101'  
**Length** \*  
**Class** CLASS  
**Sprcls** STRING - String

Description Error Code (DSCERRCD) String specifies why the target server manager is unable to assemble a valid Formatted Data Object Content Architecture (FD:OCA) descriptor object.<sup>7</sup>

clsvar	NIL	
insvar	CLASS INSTANCE VARIABLES	
length	5	
class	X'2101'	
value	INSTANCE_OF	UNSBINDR - Unsigned Binary Data Representation
	LENGTH	8
	MINVAL	1
	ENUVAL	X'01"
	NOTE	FD:OCA triplet is not used in DRDA descriptors, or the type code is invalid.
	ENUVAL	X'02'
	NOTE	FD:OCA triplet sequence error.
	ENUVAL	X'03'
	NOTE	An array description is required, and this is not one (too many or too few Row Lay Out (RLO) triplets).
	ENUVAL	X'04'
	NOTE	A row description is required, and this is not one (too many or too few RLO triplets).
	ENUVAL	X'05'
	NOTE	Late environmental descriptor just received, not supported.
	ENUVAL	X'06'
	NOTE	Malformed triplet; required parameter is missing.
	ENUVAL	X'07'
	NOTE	Parameter value is not acceptable.
	ENUVAL	X'11'
	NOTE	Meta-Data Descriptor (MDD) present is not recognized as a Structured Query Language (SQL) descriptor.

7. The FD:OCA description for the data value of this object is in the DRDA Reference.

ENUVAL	X'12'
NOTE	MDD class is not recognized as a valid SQL class.
ENUVAL	X'13'
NOTE	MDD type is not recognized as a valid SQL type.
ENUVAL	X'21'
NOTE	Representation is incompatible with SQL type (in prior MDD).
ENUVAL	X'22'
NOTE	CCSID is not supported.
ENUVAL	X'32'
NOTE	Group Data Array (GDA) references a local identifier (LID) which is not a Simple Data Array (SDA) or GDA.
ENUVAL	X'33'
NOTE	GDA length override exceeds limits.
ENUVAL	X'34'
NOTE	GDA precision exceeds limits.
ENUVAL	X'35'
NOTE	GDA scale greater than precision or scale negative.
ENUVAL	X'36'
NOTE	GDA length override missing or incompatible with data type.
ENUVAL	X'41'
NOTE	RLO references a LID which is not an RLO or GDA.
ENUVAL	X'42'
NOTE	RLO fails to reference a required GDA or RLO.

---

<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO**

<b>insvar</b>	<i>DSCINVRM</i> (on page 298)
<b>Semantic</b>	<i>DSCINVRM</i> (on page 298)

NAME

DSCINVRM — Invalid Description

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'220A'  
**Length** \*  
**Class** CLASS  
**Sprcls** RPYMSG - Reply Message

Invalid Description (DSCINVRM) Reply Message specifies that a target server manager was unable to assemble a valid Formatted Data Object Content Architecture (FD:OCA) descriptor for the data being sent. The DSCERRCD specifies the reason for the error.

This reply message indicates that the FD:OCA descriptor is invalid either because it violates FD:OCA rules or Distributed Relational Database Architecture (DRDA) rules for the construction of an FD:OCA descriptor.

The offsets the parameters FDODSCOFF, FDOTRPOFF, and FDOPRMOFF specify refer to the descriptor components that are in error.

DSS Carrier: RPYDSS

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
<b>length</b>	*	
<b>class</b>	X'220A'	
<b>dscerrcd</b>	INSTANCE_OF REQUIRED	DSCERRCD - Description Error Code
<b>fdodsc</b>	INSTANCE_OF REQUIRED	FDODSC - FD:OCA Data Descriptor
<b>fdodscoff</b>	INSTANCE_OF REQUIRED	FDODSCOFF - FD:OCA Descriptor Offset
<b>fdoprmooff</b>	INSTANCE_OF REQUIRED	FDOPRMOFF - FD:OCA Triplet Parameter Offset
<b>fdotrpooff</b>	INSTANCE_OF REQUIRED	FDOTRPOFF - FD:OCA Triplet Offset
<b>rdbnam</b>	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code  8 - ERROR - Error Severity Code
<b>clscmd</b>	<b>NIL</b>	



**inscmd**            **NIL**

**SEE ALSO**

**cmdrpy**            *EXCSQLSET* (on page 377)  
                      *EXCSQLSTT* (on page 381)  
                      *OPNQRY* (on page 555)

**Semantic**         *EXCSQLSTT* (on page 381)  
                      *OPNQRY* (on page 555)

**NAME**

DSCRDBTBL — Describe RDB Table

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2012'**Length** \***Class** CLASS**Sprcls** COMMAND - Command

Describe RDB Table (DSCRDBTBL) command requests that a description of the relational database (RDB) table named in the SQLOBJNAM command data object be returned to the requester.

**Source System Processing**

The source system determines the location of the RDB:

Local            Call the local RDB server.

Remote          Send the DSCRDBTBL command to the remote RDB server.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**DSS Carrier: RQSDSS****Target System Processing**

The *monitor* parameter allows the requester to request a set of events or activities to be monitored on the target server. Refer to MONITOR for the list of supported events.

The *rdbnam* parameter specifies the name of the RDB that the ACCRDB command accesses. If the RDBNAM parameter is specified, its value must be the same as the value specified on the ACCRDB command for RDBNAM.

The TYPDEFNAM command data object specifies the name of the data type to data representation mapping definitions that the source SQLAM uses when sending command data objects for this command.

The TYPDEFOVR command data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type to data representation mapping definitions for the command data objects for this command.

The DSCRDBTBL command must be followed by an SQLOBJNAM command data object containing the name of the RDB table being described. The SQLOBJNAM FD:OCA descriptor describes the SQLOBJNAM command data object.

Normal completion of this command results in the description of the named RDB table being returned in the SQLDARD reply data object. The SQLDARD FD:OCA descriptor describes the SQLDARD reply data object.

The SECTKNOVR reply data object is sent by the intermediate server, if the SQLDARD reply data object is encrypted. The SECTKNOVR contains the encryption seed and the encryption token used to encrypt the SQLDARD reply data object. The SECTKNOVR DSS is encrypted and must be sent prior to the encrypted SQLDARD reply data object.

The TYPDEFNAM reply data object specifies the name of the data type to data representation mapping definitions that the target SQLAM uses when sending the reply data objects for this command.

The TYPDEFOVR reply data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type to data representation mapping definitions for the reply data objects for this command.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**Exceptions**

If the current unit of work ended abnormally, an ABNUOWRM reply message followed by an SQLCARD reply data object must be returned.

Otherwise, exception conditions that the RDB detects are reported in the SQLDARD or SQLCARD object. If an SQLDARD reply data object is returned, the number of data variable definition entries contained therein must be zero. If instead an SQLCARD reply data object is returned, it can optionally be preceded by an SQLERRRM reply message.

If the bind process is active, then the command is rejected with the PKGBPARM.

If access to the specified RDB is not obtained, then the command is rejected with the RDBNACRM.

**Source System Reply Processing**

Return any reply messages and data objects to the requester.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
<b>length</b>	*	
<b>class</b>	X'2012'	
<b>monitor</b>	INSTANCE_OF OPTIONAL MINLVL	MONITOR - Monitor Events  7
<b>rdbnam</b>	INSTANCE_OF	RDBNAM - Relational Database Name

	OPTIONAL CMDTRG	
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>cmddta</b>		<b>COMMAND OBJECTS</b>
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL MINLVL ENUVAL MINLVL OPTIONAL DFTVAL NOTE	TYPDEFNAM - Data Type Definition Name 'QTDSQL370' 'QTDSQL400' 'QTDSQLX86' 'QTDSQLASC' 4 'QTDSQLVAX' 4 " The default means the value received on ACCRDB command is used.
X'0035'	INSTANCE_OF OPTIONAL DFTVAL NOTE	TYPDEFOVR - TYPDEF Overrides " The default means the value received on ACCRDB command is used.
X'243E'	INSTANCE_OF REQUIRED	SQLOBJNAM - SQL Object Name
<b>rpydta</b>		<b>REPLY OBJECTS</b>
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL MINLVL ENUVAL MINLVL OPTIONAL DFTVAL NOTE	TYPDEFNAM - Data Type Definition Name 'QTDSQL370' 'QTDSQL400' 'QTDSQLX86' 'QTDSQLASC' 4 'QTDSQLVAX' 4 " The default means the value received on ACCRDBRM is used.
X'0035'	INSTANCE_OF OPTIONAL DFTVAL NOTE	TYPDEFOVR - TYPDEF Overrides " The default means the value received on ACCRDBRM is used.
X'190B'	INSTANCE_OF OPTIONAL NOTE	SECTKNOVR - SECTKN Override  This is sent by the intermediate server, if the SQLDARD reply data object is encrypted. This must precede the SQLDARD reply data object.

X'1C00'	INSTANCE_OF OPTIONAL MINLVL	MONITORRD - Monitor Reply Data 7
X'1C03'	INSTANCE_OF OPTIONAL MINLVL	MGRLVLOVR - Manager Level Overrides 7
X'2411'	INSTANCE_OF REQUIRED MTLEXC	SQLDARD - SQLDA Reply Data X'2408' - SQLCARD - SQL Communications Area Reply Data
X'2408'	INSTANCE_OF MTLEXC	SQLCARD - SQL Communications Area Reply Data X'2411' - SQLDARD - SQLDA Reply Data
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'220D'	INSTANCE_OF	ABNUOWRM - Abnormal End Unit of Work Condition
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported
X'2209'	INSTANCE_OF	PKGBPARM - RDB Package Binding Process Active
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'2204'	INSTANCE_OF	RDBNACRM - RDB Not Accessed
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limits Reached
X'2213'	INSTANCE_OF	SQLERRRM - SQL Error Condition
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'125F'	INSTANCE_OF	TRGNSPRM - Target Not Supported
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

**SEE ALSO**

<b>inscmd</b>	<i>SQLAM</i> (on page 847)
<b>Semantic</b>	<i>BGNBND</i> (on page 110) <i>SECTKNOVR</i> (on page 822) <i>SQL</i> (on page 835) <i>SQLAM</i> (on page 847)

## NAME

DSCSQLSTT — Describe SQL Statement

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD

**Codepoint** X'2008'

**Length** \*

**Class** CLASS

**Sprcls** COMMAND - Command

The Describe SQL Statement (DSCSQLSTT) command returns the definitions of either the select list or the input data variables referenced within the specified package section.

**Source System Processing**

The source system determines the location of the RDB:

Local Call the local RDB server.

Remote Send the DSCSQLSTT command to the remote RDB server.

**DSS Carrier: RQSDSS****Target System Processing**

The optional *cmdsrid* parameter uniquely identifies the source of the command.

The *monitor* parameter allows the requester to request a set of events or activities to be monitored on the target server. Refer to *MONITOR* (on page 521) for the list of supported events.

The *pkgnamcsn* parameter specifies the fully qualified package name, the package's consistency token, and a section number within the package of the Structured Query Language (SQL) statement whose select list column definitions are being returned.

The *qryinsid* parameter specifies the instance of the query in use. This parameter is required if the SQL statement being described has an open cursor associated with it. Its value must match the *qryinsid* parameter returned on the OPNQRYRM reply message for this instance of the query.

The *rdbnam* parameter specifies the name of the relational database (RDB) to which the ACCRDB command gained access. If the *rdbnam* parameter is specified, its value must be the same as the value specified on the ACCRDB command for RDBNAM.

Normal completion of this command results in either the select list data variable definitions or the input data variable definitions being returned in the SQLDARD reply data object. The SQLDARD FD:OCA descriptor describes the SQLDARD reply data object.

The SECTKNOVR reply data object is sent by the intermediate server, if the SQLDARD reply data object is encrypted. The SECTKNOVR contains the encryption seed and the encryption token used to encrypt the SQLDARD reply data object. The SECTKNOVR DSS is encrypted and must be sent prior to the encrypted SQLDARD reply data object.

The TYPDEFNAM reply data object specifies the name of the data type to data representation mapping definitions that the target SQLAM uses when sending the reply data objects for this command.

The TYPDEFOVR reply data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type to data representation mapping definitions for the reply data objects for this command.

The TYPSQLDA instance variable determines the type of descriptive information to be returned for input or output variables. Three types of input or output SQLDA descriptor area can be requested: a light, standard, or extended version. Each type provides a different set of descriptive information. The light SQLDA provides minimal descriptive information. The standard SQLDA provides the same descriptive information as was defined for SQLAM level 6 or earlier. The extended SQLDA provides additional descriptive information required by certain types of API such as JDBC.

The SQLDA types are defined by the SQLDARD FD:OCA descriptor. Refer to the DRDA Reference for a description of an SQLDARD.

### Data Conversion

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

### Exceptions

If the current unit of work ended abnormally, an ABNUOWRM reply message followed by an SQLCARD reply data object must be returned.

Otherwise, exception conditions that the RDB detects are reported in the SQLDARD or SQLCARD object. If an SQLDARD reply data object is returned, the number of data variable definition entries contained therein must be zero. If instead in SQLCARD reply data object is returned, it can optionally be preceded by an SQLERRRM reply message.

If the bind process is active, then the command is rejected with the PKGBPARM.

If the specified RDB is not currently accessed, then the command is rejected with the RDBNACRM.

### Source System Reply Processing

Return any reply messages and data objects to the requester.

### Package Name and Consistency Token

With SQLAM level 7, the PKGNAMCSN is changed to an optional instance variable. If specified, the values are used as the default package name and consistency token for the connection. If not specified on the command, the PKGSN is required to identify the section number. In this case, the default package name, consistency token, and specified section number is used for the request. If PKGNAMCSN was never specified on the connection, a PRCCNVRM is returned.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>INSTANCE VARIABLES</b>
<b>length</b>	<b>*</b>
<b>class</b>	<b>X'2008'</b>

cmdsrcid	INSTANCE_OF OPTIONAL MINLVL	CMDSRCID - Command Source Identifier 7
monitor	INSTANCE_OF OPTIONAL MINLVL	MONITOR - Monitor Events 7
pkgnamcsn	INSTANCE_OF MINLVL OPTIONAL MTLEXC	PKGNAMECSN - RDB Package Name, Consistency Token, and Section Number 7 Required if PKGSN not specified PKGSN
pkgsn	INSTANCE_OF MINLVL OPTIONAL MTLEXC	PKGSN - RDB Package Section Number 7 Required if PKGNAMCSN not specified. PKGNAMCSN
qryinsid	INSTANCE_OF REQUIRED  MINLVL	QRYINSID - Query Instance Identifier If the DSCSQLSTT command is operating on an SQL statement that has an open cursor associated with it. 7
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
typsqlda	INSTANCE_OF  MINLVL OPTIONAL	TYPSQLDA - Type of SQL Descriptor Area <QDDRDBD> 6
clscmd	NIL	
inscmd	NIL	
cmddta	NIL	
<b>rpydta</b>		<b>REPLY OBJECTS</b>
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL MINLVL ENUVAL MINLVL OPTIONAL DFTVAL NOTE	TYPDEFNAM - Data Type Definition Name 'QTDSQL370' 'QTDSQL400' 'QTDSQLX86' 'QTDSQLASC' 4 'QTDSQLVAX' 4 " The default means the value received on ACCRDBRM is used.
X'0035'	INSTANCE_OF OPTIONAL DFTVAL NOTE	TYPDEFOVR - TYPDEF Overrides " The default means the value received on ACCRDBRM is used.



X'190B'	INSTANCE_OF OPTIONAL NOTE	SECTKNOVR - SECTKN Override  This is sent by the intermediate server, if the SQLDARD reply data object is encrypted. This must precede the SQLDARD reply data object.
X'1C00'	INSTANCE_OF OPTIONAL MINLVL	MONITORRD - Monitor Reply Data  7
X'1C03'	INSTANCE_OF OPTIONAL MINLVL	MGRVLVLOVR - Manager Level Overrides  7
X'2411'	INSTANCE_OF REQUIRED MTLEXC	SQLDARD - SQLDA Reply Data  X'2408' - SQLCARD - SQL Communications Area Reply Data
X'2408'	INSTANCE_OF  MTLEXC	SQLCARD - SQL Communications Area Reply Data X'2411' - SQLDARD - SQLDA Reply Data
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'220D'	INSTANCE_OF	ABNUOWRM - Abnormal End Unit of Work Condition
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported
X'2209'	INSTANCE_OF	PKGBPARAM - RDB Package Binding Process Active
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'2204'	INSTANCE_OF	RDBNACRM - RDB Not Accessed
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limits Reached
X'2213'	INSTANCE_OF	SQLERRRM - SQL Error Condition
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'125F'	INSTANCE_OF	TRGNSPRM - Target Not Supported
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

**SEE ALSO**

<b>inscmd</b>	<i>SQLAM</i> (on page 847)
<b>Semantic</b>	<i>BGNBND</i> (on page 110) <i>CMDSRCID</i> (on page 178) <i>SECTKNOVR</i> (on page 822) <i>SQL</i> (on page 835) <i>SQLAM</i> (on page 847)

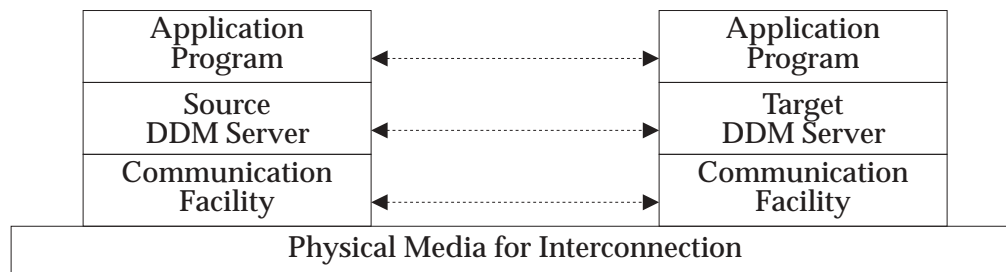
## NAME

DSS — Data Stream Structures

## DESCRIPTION (Semantic)

**Dictionary** QDDTTRD**Length** \***Class** CLASS**Sprcls** DATA - Encoded Information

Data Stream Structures (DSS) are how the DDM architecture views each communication facility, as an ordered set of layers. Figure 3-24 is an illustration of layered systems. The number of layers comprising the communication facility depends on the communication facility. Each communication facility provides services that must be communicated through the data stream structures defined for that layer.



**Figure 3-24** Layered Communications Reference Model

As in other layered communication models, communication logically takes place between each layer while supporting the requirements of the layer above it. The communications facilities handle real communications. The application program on the source system requests DDM services through a programming interface that the source system defines. No interface between a target system DDM and an application program has been defined. Thus, there are no direct program-to-program communications.

DDM is viewed as a multi-layer architecture for communicating data management requests between servers located on different systems. All information is exchanged in the form of objects which are mapped onto a data stream appropriate to the communication facilities DDM uses.

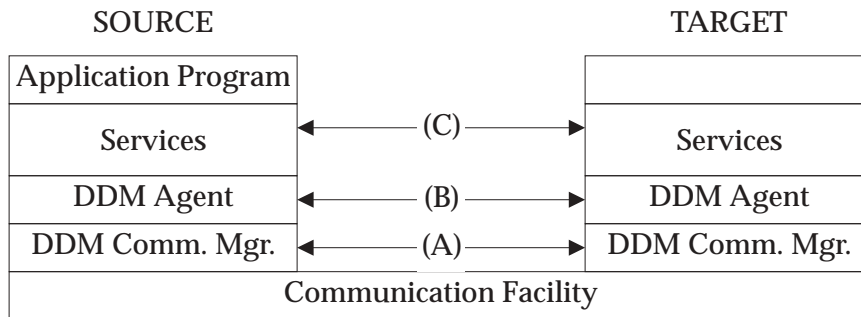
DDM architecture has the following three distinct layers, as shown in Figure 3-25 (on page 309):

**Layer A** Communications management services

**Layer B** Agent services

**Layer C** Data management services

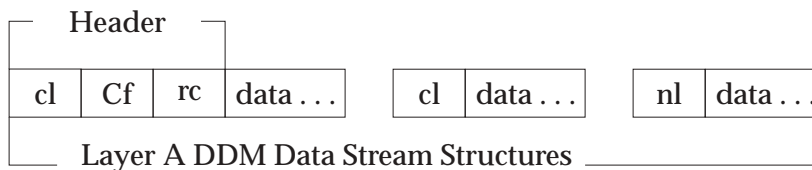
The levels clearly separate management from the communications facilities and separate flow protocols (A) from the parsing of DDM messages (B) and from data management services (C).



**Figure 3-25** Layers (A, B, and C) in the DDM Architecture

**SNA LU 6.2 Communication Facility**

The DDM layers reside on top of the SNA LU 6.2 communication facility consisting of five layers. SNA Layer 5 must ensure that a two-byte length field precedes each independently transmitted structure. The high-order bit of this length field controls whether the content of the structure is continued in the next structure. DDM also meets the requirements of Systems Network Architecture (SNA) at this layer by following the length field with a two-byte identifier registered with SNA (see *DDMID* (on page 261)). This is illustrated in Figure 3-26.



**Figure 3-26** DDM Layer A Data Stream Structure

**Legend**

- cl** Two-byte length field with continuation flag set ON.
- nl** Two-byte length field with continuation flag set OFF.
- C** One-byte DDM identifier (D0 for all DDM data).
- f** One-byte qualifier of the C-byte (used as a format identifier by DDM).
- rc** Request correlation identifier.

**TCP/IP Communication Facility**

The DDM envelope to send data via TCP/IP requires a six byte header. The first two bytes are designated for length of data with continuation flag set ON. The third byte is Hex 'D0' to indicate DDM data. The fourth byte is the DDM format byte. The last two bytes are the request correlation identifier. This is illustrated in Figure 3-26.

**Layer A**

At Layer A, the DDM communications managers provide these services:

- Request, reply, and data correlation
- Structure chaining to reduce communication flows  
See Figure 3-27 (on page 311) and Figure 3-28 (on page 312) for DSS chaining rules.
- Continuation or termination of chains when errors are detected  
See Figure 3-28 (on page 312) for DSS chain termination.
- Interleaving and multi-leaving request, reply, and data DSSs for multitasking environments
- Communications messages are used between the communication managers for controlling multitasking environments.

The data stream structures Layer A envelopes are illustrated in Figure 3-30 (on page 313) and Figure 3-31 (on page 314).

The *f* byte qualifier is used to specify (see *DSSFMT* (on page 318)):

- Whether the structure is a request structure, a reply structure, or an object structure and, if a request structure, whether or not a reply DSS or reply OBJDSS is expected from the command and, if an object structure, whether or not OBJDSS is encrypted.
- Whether a structure is chained to a subsequent structure.

Chaining allows a sequence of request and object structures (RQSDSS chain) or reply and object structures (RPYDSS chain) to be transmitted as a unit to improve performance.

- Whether the next request in a chain is processed when the current request returns a reply message with a severity code value of ERROR or greater.

Error continuation can be terminated under certain conditions. If the command terminates with one of the following reply messages having a severity code value of ERROR or greater, then error continuation can be terminated.

The target server must perform error continuation, when requested, unless one of these reply messages is sent:

AGNPRMRM	Permanent Agent Error
CMDCHKRM	Command Check
RSCLMTRM	Resource Limits Reached
PRCCNVRM	Conversational Protocol Error
SYNTAXRM	Data Stream Syntax Error
ABNUOWRM	Abnormal End Unit of Work Condition

- Whether the next DSS in a chain has the same or different request correlator from the current DSS.

A communications manager can determine whether it has received all the replies, reply data objects, or command data objects related to a single command without having to receive a DSS related to a different command.

- Whether the DSS is a communications message between the communications managers.

The high-order part of the *f* sets the execution priority of the receiving agent.

A two-byte identifier is the last part of the DDM header to support request correlation. This identifier correlates requests with their associated objects, replies, and reply objects. Each request in a chain must be assigned a unique correlation identifier. For example, in the conversational protocol, the sender can optionally restart at zero for each chain. See *RQSCR* (on page 772) for a discussion of correlation.

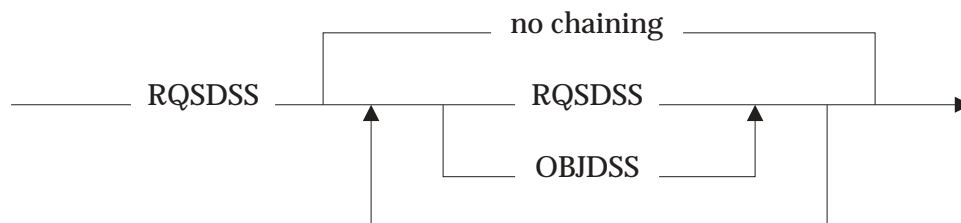
Three general types of DDM data stream structures are mapped onto the DDM header structure:

1. Request structures are used for all requests to a target server agent for processing (see *RQSDSS* (on page 774)).
2. Reply structures are used for all replies from a target agent to a source agent regarding conditions detected during the processing of a request (see *RPYDSS* (on page 766)).
3. Object structures are used for all objects (for instance, records) sent between DDM Agents (see *OBJDSS* (on page 536)).

### DSS Chaining

The following sections describe the DSS chaining rules for the transmissions from source to target and target to source.

The NOOPR command allows the source communications manager to end a chain after the last element of a chain has been processed.



**Figure 3-27** RQSDSS Chaining Rules

### RQSDSS Chaining Rules

The following rules apply to Figure 3-27:

1. The first DSS in the chain must be an RQSDSS.
2. If this RQSDSS is the only DSS in the chain, no chaining is done.
3. If data is sent with the RQSDSS, it must immediately follow the RQSDSS in the DSS chain.
4. All DSSs with the same correlation identifier must be contiguous in the chain.
5. The next DSS can be an RQSDSS or an OBJDSS.
6. Repeat steps 3 through 5 as needed.

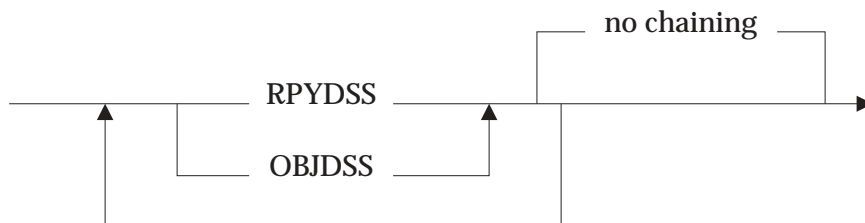


Figure 3-28 RPYDSS and OBJDSS Chaining Rules

**RPYDSS and OBJDSS Chaining Rules**

The following rules apply to Figure 3-28:

1. All DSSs with the same correlation identifier must be contiguous in the chain.
2. If only one RPYDSS or OBJDSS is in the chain, no chaining is done.

RQSDSS Chain Status	RQSDSS DSSs	RPYDSS Chain		
		DSSs	Severity	
done	RQSDSS 1	RPYDSS 1	1	0
done	RQSDSS 2	OBJDSS	2	
		RPYDSS 2	4	
in process	RQSDSS 3	RPYDSS 3	3	16
— RQSDSS chain terminated—				
	RQSDSS 4			
	RQSDSS 5			

Figure 3-29 RQSDSS Chain Error Termination

**RQSDSS Chain Error Termination**

The following rules apply to Figure 3-29:

1. The error continuation format bit in all of the RQSDSSs specified continuation on ERROR (severity code > 8).
2. If the error continuation format bit of RQSDSS 3 has been set to *do not continue on error*, then the chain would be terminated after RQSDSS 3. *do not continue on error* means to terminate on a severity code of ERROR (8) or greater.
3. RPYDSS 3 contains a reply message with severity code 8.
4. Processing of the RQSDSS chain is terminated, and RQSDSSs 4 and 5 are discarded without processing.
5. The RPYDSS chain is sent to the source server.

Rules for the *continue on error* are:

- If the bit is set to zero, a chain is broken when the severity code of a reply message is ERROR (8) or greater.
- If the bit is set to one, a chain is broken when the severity code of a reply message is SEVERE (16) or greater, or one of the following reply messages is used: AGNPRMRM, CMDCHKRM, RSCLMTRM, PRCCNVRM, SYNTAXRM, or ABNUOWRM.

**Layer B**

All data stream structures at Layer B are formally defined and uniformly modeled as objects. Layer B data stream structures (DSSs) carry objects. A single DSS always carries a single object. A single DSS can also carry multiple whole objects. In either case, the DSS can be segmented at SNA Layer 5 as shown in Figure 3-30. At Layer B, the primary data stream services DDM agents provide are:

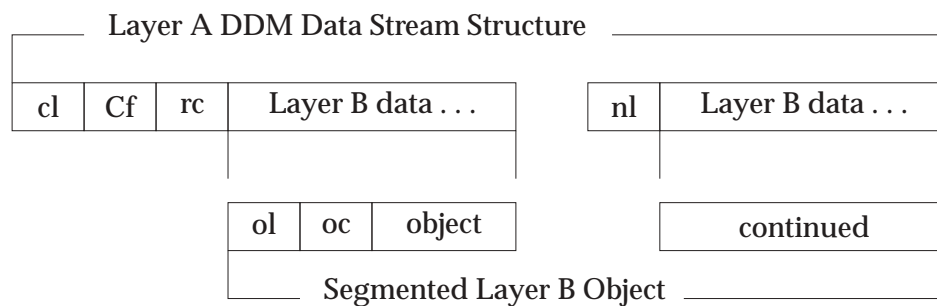
- To map objects from a server-specific memory form to a canonical data stream form.
- To map objects from the canonical data stream form to a server-specific memory form.
- To validate objects received.
- To route commands to target programs.

Layer B objects with data 5 bytes less than 32K bytes (32,763 bytes) long consist of:

- A two-byte length field
- A two-byte class field that identifies the class (type) of an object by its codepoint
- Object data

This is either SCALAR data or COLLECTION data. Scalar data consists of a string of bytes formatted as the class description for the object required. Collections consist of a set of objects in which the entries in the collection are nested within the length/code point of the collection.

These objects are mapped onto the data portion of the Layer A data stream structures as shown in Figure 3-30. Note that the object can be segmented at Layer A to accommodate transmission buffer restrictions (or for any other reason).



**Figure 3-30** Mapping Small DDM Layer B Objects to Layer A DSSs

**Legend**

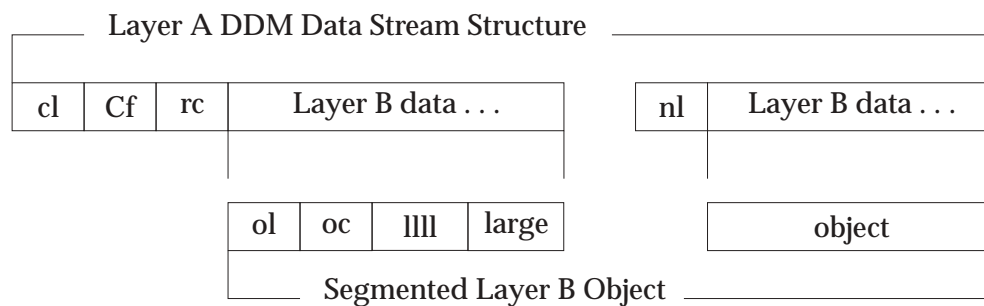
- cl** Two-byte length field with continuation flag set ON.
- nl** Two-byte length field with continuation flag set OFF.
- C** One-byte DDM identifier (D0 for all DDM data).
- f** One-byte qualifier of the C-byte (used as a format identifier by DDM).
- rc** Request correlation identifier.
- ol** Two-byte object length field.

**oc** Two-byte object codepoint.

Most DDM objects are small, but for objects with data greater than 32 Kbytes less 5 bytes (32,763 bytes) long an extended total length field must augment the normal two-byte object length. Setting the high-order bit of the object length field to 1 specifies the following:

- An "extended total length" field immediately follows the "class" field of the object and precedes its data.
- The object length field does not include the length of the object's data. It includes only the length of the class, length, and extended total length fields.
- The value of the extended total length field specifies only the length of the object's data. It does not include its own length.
- The length of the extended total length field must be a multiple of two bytes, and it cannot be longer than necessary to express the length of the object's data.

Large objects are mapped onto the data portion of the Layer A data stream structures as shown in Figure 3-31. Note that the object can be segmented at Layer A to accommodate transmission buffer restrictions (or for any other reason).



**Figure 3-31** Mapping Large DDM Layer B Objects to Layer A DSSs

### Legend

- cl** Two-byte length field with continuation flag set ON.
- nl** Two-byte length field with continuation flag set OFF.
- C** One-byte DDM identifier (D0 for all DDM data).
- f** One-byte qualifier of the C-byte (used as a format identifier by DDM).
- rc** Request correlation identifier.
- ol** Two-byte object length field.
- oc** Two-byte object codepoint.
- llll** Extended total length field.



### DSS Encryption

The Layer B object is encrypted for DDM data stream structure type, Encrypted Object DSS.

The encrypted Layer B objects consist of:

- Layer B header
- Object data

For Encrypted Object DSS, the Layer A DSS length can increase up to 8 bytes after encryption and Layer A DSS length can decrease up to 8 bytes after decryption.

For Encrypted Object DSS, if the object is segmented at Layer A to accommodate transmission buffer restrictions (or for any other reason), then the segmented object must be combined and decrypted as a single object.

### Layer B Streaming

A Layer B object can be sent without its 2-byte length field reflecting its actual length using a technique called *streaming*. This mechanism is most useful for the sender of a Layer B object when the object length is unknown at the time its header is generated, and it is not practical for the sender to determine the length. However, the sender can choose to always stream even when the Layer B object length is known. The following applies for streaming:

- The DSS must be an OBJDSS.
- There is only one Layer B object which must be either an EXTDTA or QRYDTA within the OBJDSS.
- The sender (source or target server) sets the Layer B 2-byte object length field (ol in Figure 3-30 (on page 313) and Figure 3-31 (on page 314) above) to X'8004'.
- The Layer B extended total length field (llll in Figure 3-31 (on page 314) above) is never present regardless of the object length.
- While streaming is optional for the source or target server as a sender, the receiver (source or target server) must support it.
- The receiver (source or target server) determines the length of the Layer B object (not including its header) as follows:

$$\text{Object length} = ((n-1) * 32,767 + n1) - ((6+4 + ((n-1) * 2)))$$

where:

- $n$  = Total number of DSS segments used
- $32,767$  = Length of full DSS segment
- $n1$  = Length of last DSS segment
- $6$  = Length of Layer A header
- $4$  = Length of Layer B header
- $2$  = Length of Layer A length field

**Layer C**

At Layer C, the services each class of DDM object (such as sequential files) provides are defined by:

- Specific commands and objects that can be sent to instances of the class
- Reply messages and objects that instances of the class can return in response to the commands

All commands are defined as subclasses of COMMAND and are carried by an RQSDSS. An RQSDSS can carry only one command.

All reply messages are defined as subclasses of RPYMSG and are carried by an RPYDSS. The same RPYDSS can carry one or more reply messages.

All command data objects and reply data objects, which an OBJDSS carries, have superclass chains of SCALAR or COLLECTION and then OBJECT. The same OBJDSS can carry one or more objects.

The general rule for building data stream structures is to build a structure as specified for instances of the class and to refer to the classes it references. A reference to a constant as a value means *code the constant data as specified*. A reference to a CLASS means, *code an instance of that class as described by its class*. A summary of the DDM Data Stream Structure and the layers is illustrated in Figure 3-32.

Layer C	Specific Commands and their Parameters	Specific Replies and their Parameters	Specific Scalars and Collections	
Layer B	Commands	Reply Messages	Scalars and Collections	Communications
Layer A	RQSDSS	RPYDSS	OBJDSS	CMNDSS Mapped Data
DDM Data Stream Structures				

**Figure 3-32** Summary of DDM Data Stream Structure Layers

**SEE ALSO**

- insvar**      *DSSFMT* (on page 318)  
                   *OBJDSS* (on page 536)  
                   *PRCCNVCD* (on page 621)  
                   *RPYDSS* (on page 766)  
                   *RQSDSS* (on page 774)  
                   *SYNERRCD* (on page 985)
- Semantic**    *AGENT* (on page 61)  
                   *APPSRCER* (on page 91)  
                   *CMNSYNCPT* (on page 202)  
                   *COLLECTION* (on page 238)  
                   *CONCEPTS* (on page 243)  
                   *DATA* (on page 251)  
                   *DSSFMT* (on page 318)

*OBJDSS* (on page 536)  
*QRYBLK* (on page 670)  
*RDBOVR* (on page 740)  
*RPYDSS* (on page 766)  
*RQSDSS* (on page 774)  
*RSYNCMGR* (on page 787)  
*SCALAR* (on page 796)  
*SYNCPTMGR* (on page 939)  
*SYNTAXRM* (on page 989)  
*TCPSRCER* (on page 1013)

NAME

DSSFMT — Data Stream Structure Format

DESCRIPTION (Semantic)

**Dictionary** QDDBASD

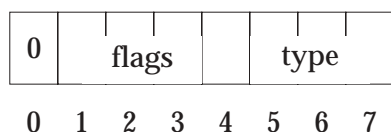
**Codepoint** X'140D'

**Length** \*

**Class** CLASS

**Sprcls** HEXSTRDR - Hexadecimal String

Data Stream Structure Format (DSSFMT) Hexadecimal String specifies the format of a DDM data stream structure (DSS). It specifies the type of the DSS and how it relates to other DSSs. Figure 3-33 illustrates the *format byte* layout for a DSS.



flags = structure-chained flag and continue-on-error flag  
 type = command, reply, or object type DSS

**Figure 3-33** DSS Format Byte Layout

*flags* Structure-encrypted flag, structure chained flag, and continue-on-error flag.

*type* Command, reply, or object type DSS.

Bits 1-3 are the flags. Bits 4-7 are the type.

clsvar	NIL	
insvar	CLASS INSTANCE VARIABLES	
bit1	INSTANCE_OF	BITDR - A Single Bit
	LENGTH	1
	ENUVAL	B'0'
	NOTE	Structure not chained to next structure.
	ENUVAL	B'1'
	NOTE	Structure chained to next structure.
bit2	INSTANCE_OF	BITDR - A Single Bit
	LENGTH	1
	ENUVAL	B'0'
	NOTE	Do not continue on error.
	ENUVAL	B'1'
	NOTE	Continue on error.
	NOTE	Must be B'0' if the chaining bit (bit1) is B'0'.
bit3	INSTANCE_OF	BITDR - A Single Bit
	LENGTH	1

	ENUVAL	B'0'
	NOTE	Next DSS has different request correlator.
	ENUVAL	B'1'
	NOTE	Next DSS has same request correlator.
	NOTE	Must be B'0' if the chaining bit (bit1) is B'0'.
<hr/>		
dsstyp	INSTANCE_OF	HEXDR - Hexadecimal Number
	LENGTH	1
	ENUVAL	1
	NOTE	Request DSS.
	ENUVAL	2
	NOTE	Reply DSS.
	ENUVAL	3
	NOTE	Object DSS.
	ENUVAL	4
	NOTE	Encrypted Object DSS.
	MINLVL	5
	ENUVAL	5
	NOTE	Request DSS where no reply is expected.
	MINLVL	5
<hr/>		
cls cmd	NIL	
<hr/>		
ins cmd	NIL	
<hr/>		
vldattls	NIL	

**SEE ALSO**

<b>insvar</b>	<i>OBJDSS</i> (on page 536) <i>RPYDSS</i> (on page 766) <i>RQSDSS</i> (on page 774) <i>SYNERRCD</i> (on page 985)
<b>Semantic</b>	<i>BGNATMCHN</i> (on page 107) <i>DSS</i> (on page 308) <i>SYNCPTOV</i> (on page 944)

## NAME

DTAMCHRM — Data Descriptor Mismatch

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD**Codepoint** X'220E'**Length** \***Class** CLASS**Sprcls** RPYMSG - Reply Message

Data Descriptor Mismatch (DTAMCHRM) Reply Message indicates that:

- The descriptor received did not violate any Formatted Data Object Content Architecture (FD:OCA) or Distributed Relational Database Architecture (DRDA) rules and was successfully assembled.
- The data received did not match the received descriptor. That is, the amount of data received did not match the amount of data expected.

**DSS Carrier:** RPYDSS

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'220E'	
<b>rdbnam</b>	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

## SEE ALSO

**cmdrpy** *BNDSQLSTT* (on page 136)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSET* (on page 377)  
*EXCSQLSTT* (on page 381)  
*OPNQRY* (on page 555)  
*PRPSQLSTT* (on page 636)

**Semantic** *BNDSQLSTT* (on page 136)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSET* (on page 377)  
*EXCSQLSTT* (on page 381)  
*OPNQRY* (on page 555)  
*PRPSQLSTT* (on page 636)

**NAME**

DTAOVR — Data Layer Overview

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

Data Layer Overview (DTAOVR) discusses data layers which consist of the data values of:

- Architected scalar objects, such as names and attributes.

The scalar object classes of the DDM architecture describe these data values.

- The DDM architecture includes class objects describing the representations of a wide range of data types.

These classes all have names of the form xxxDR, such as CHRDR and BINDR. These data classes and their attributes describe both user data and the objects the DDM architecture defines.

- Relational database rows.

The data definition facilities of the Structured Query Language (SQL) describe these data values and store them within a relational database (RDB). The DDM architecture does not describe these data values.

**SEE ALSO****Semantic** *DDM* (on page 260)

**NAME**

DUPQRYOK — Duplicate Query Allowed

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'210B'  
**Length** \*  
**Class** CLASS  
**Sprcls** BOOLEAN - Logical Value

Duplicate Query Allowed (DUPQRYOK) is specified by a source server on the OPNQRY command to indicate to the target server whether it should allow opening a query for a duplicate cursor. For the definition of a duplicate cursor, refer to Section 4.4.6 of the DRDA Reference.

If the target server is requested to open a query for a duplicate cursor but is not allowed to do so as per the setting of DUPQRYOK, it must return a QRYPOPRM reply message in accordance with the DRDA Query Instance (QI) rules as defined in the DRDA Reference.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	5	
class	X'210B'	
value	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	MINLVL	7
	LENGTH	2
	ENUVAL	X'F1' - TRUE - True State
	NOTE	Allow opening a query for a duplicate cursor.
	ENUVAL	X'F0' - FALSE - False State
	NOTE	Do not allow opening a query for a duplicate cursor.
	DFTVAL	X'F0' - FALSE - False State
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** OPNQRY (on page 555)



**NAME**

EDTASECOVR — Encrypted Data Security Overview

**DESCRIPTION (Semantic)**

**Dictionary** QDDTTRD

**Length** \*

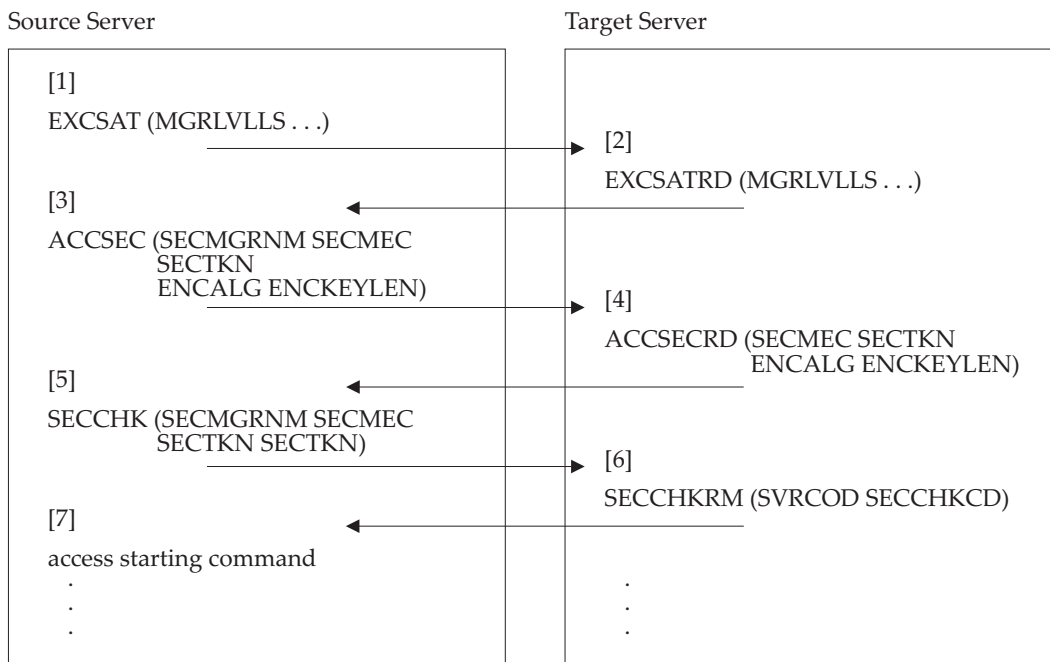
**Class** HELP

The Encrypted Data Security Overview (EDTASECOVR) provides an overview of the security-sensitive data encryption security mechanisms. The security-sensitive objects that are encrypted are SQLDTA, SQLDTARD, SQLSTT, SQLSTTVRB, SQLATTR, SQLDARD, SQLCINRD, SQLRSLRD, QRYDTA, EXTDTA, and SECTKNOVR.

**Encrypted User ID, Password, and Security-Sensitive Data Security**

One method to avoid sending the user ID, password, and security-sensitive data in clear is to encrypt the user ID, password, and security-sensitive data.

Figure 3-34 indicates the DDM commands and replies that flow in the normal process of establishing a connection while using the Encrypted User ID, Password, and Security-sensitive Data security mechanism for identification and authentication.



**Figure 3-34** DDM Encrypted User ID Password Data Security Flows

The following is a brief description of some of the parameters for the DDM commands. See the appropriate terms for a detailed description of the parameters.

1. The source server identifies the level of security it would like to operate at by identifying the SECMGR at manager level 7 on the EXCSAT command.

```

EXCSAT (
    MGRLVLS (

```

```

MGRLVL (SECMGR, 7)
.
.
.))

```

2. The target 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 target server sends the EXCSATRD reply data to the source server.

```

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

```

3. The source server receives the EXCSATRD reply data. The source server specifies the EUSRPWDDTA security mechanism combination for authentication processing on the ACCSEC command.

The SECMEC parameter indicates the security mechanism combination to use. This example assumes that the encrypted user ID, password, and security-sensitive data security mechanism are specified. The SECTKN instance variable contains the requester connection key.

The ENCALG parameter indicates the encryption algorithm to use. This example assumes that the default DES encryption security algorithm is specified. The ENCKEYLEN parameter indicates the encryption key length to use. This example assumes that the default 56-bit encryption is specified.

4. The target server receives the ACCSEC command. It supports the security mechanism identified in the SECMEC parameter, encryption algorithm specified in the ENCALG parameter, and encryption key length specified in the ENCKEYLEN parameter. The target server returns the SECMEC parameter, ENCALG parameter, and ENCKEYLEN parameter matching the source server in the ACCSECRD parameter. The target server returns the server connection key in the SECTKN parameter.

If the target server does not support or accept the security mechanism specified in the SECMEC parameter, then the target server returns the security mechanism combination values that it supports in the SECMEC parameter on the ACCSECRD object. If the target server supports the security mechanism and does not support or accept the encryption algorithm specified in the ENCALG parameter, then the target server returns the encryption algorithm combination values that it supports for the requested security mechanism in the ENCALG parameter on the ACCSECRD object. If the target server supports the security mechanism and does not support or accept the encryption key length specified in the ENCKEYLEN parameter, then the target server returns the encryption key length combination values that it supports for the requested security mechanism in the ENCKEYLEN parameter in the ACCSECRD object.

If the security token on ACCSEC is missing or if it is trivial (all binary zeros), the target server returns SECCHKCD with ACCSECRD.

5. The source server receives the ACCSECRD object and generates the encrypted security tokens containing the security context information required for security processing.

The actual process to generate the security context information is not specified by DDM. The source server may either generate the security context information, or it may call a

security resource manager to generate the security context information.

The source server passes the security context information in a SECCHK command with two SECTKN objects.

6. The target server receives the SECCHK and SECTKNs. The target server decrypts the encrypted security tokens and uses the values to perform end-user authentication and other security checks.

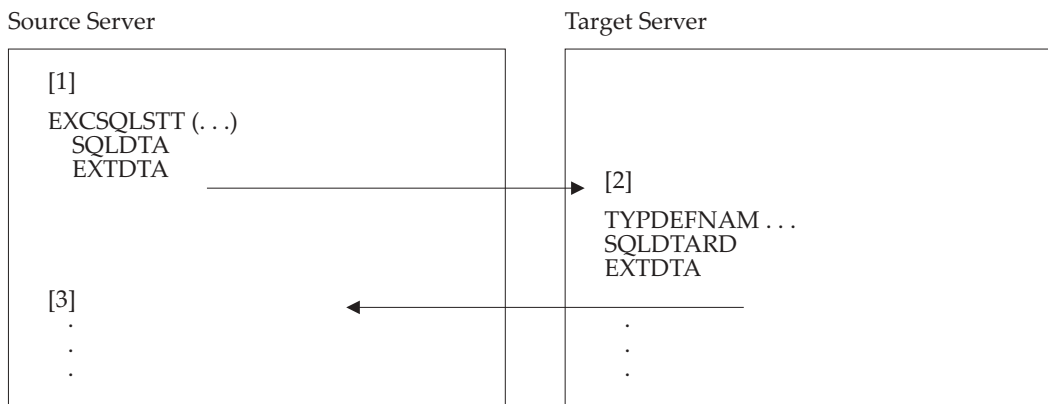
The actual process to verify the security context information is not specified by DDM. The target server may either process the security context information itself or it may call a security resource manager to process the security context information.

The target server generates a SECCHKRM to return to the source server. 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.

7. The source server receives the SECCHKRM. Assuming security processing is successful, the source server sends a data access starting command to the target server.

If security processing fails, the source server might attempt recovery before returning to the application. If the error condition is not recoverable, the source server returns to the application with an error indicating a security verification failure.

Figure 3-35 shows an example of the DDM commands and replies that flow when security-sensitive data is encrypted using the Encrypted User ID, Password, and Security-sensitive Data security mechanism.



**Figure 3-35** DDM Encrypted User ID, Password, and Security-Sensitive Data

The following is a brief description of some of the parameters for the DDM commands. See the appropriate terms for a detailed description of the parameters.

1. The source server generates an EXCSQLSTT command with encrypted SQLDTA and encrypted EXTDTA. The source server encrypts the DSS carrier for the SQLDTA and EXTDTA FD:OCA objects. The source server passes the encrypted SQLDTA and encrypted EXTDTA with the EXCSQLSTT command.

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

2. The target server receives the EXCSQLSTT with encrypted SQLDTA and encrypted EXTDTA. The target server decrypts the encrypted objects and uses the values for SQL

statement execution.

The target server generates a reply for the EXCSQLSTT command with encrypted SQLDTARD and encrypted EXTDTA. The target server encrypts the DSS carrier for the SQLDTARD and EXTDTA FD:OCA objects. The target server returns the encrypted SQLDTARD and encrypted EXTDTA to the source server.

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

3. The source server receives encrypted SQLDTARD and encrypted EXTDTA. The source server decrypts the encrypted objects.

Figure 3-36 shows an example of the DDM commands and replies that flow when security-sensitive data is encrypted using the Encrypted User ID, Password, and Security-sensitive Data security mechanism that involves intermediate server processing.

The flows described below are optional for intermediate server processing, when the intermediate server chooses not to decrypt and re-encrypt the security-sensitive data, by the use of SECTKNOVR.

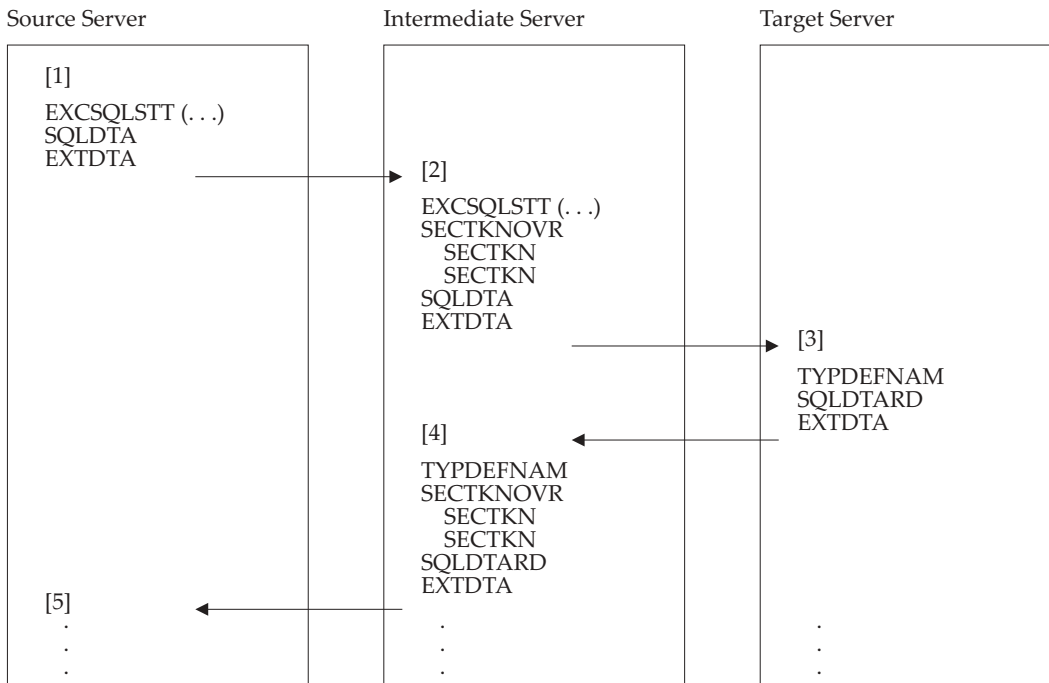


Figure 3-36 Data Encryption Flow for Intermediate Server

The following is a brief description of some of the parameters for the DDM commands. See the appropriate terms for a detailed description of the parameters.

1. The source server generates an EXCSQLSTT command with encrypted SQLDTA and encrypted EXTDTA. The source passes the encrypted SQLDTA and encrypted EXTDTA with the EXCSQLSTT command.
2. The intermediate server receives the EXCSQLSTT with encrypted SQLDTA and encrypted EXTDTA. The intermediate server passes the two SECTKN objects used for encrypting the security-sensitive objects in SECTKNOVR and encrypts the SECTKNOVR DSS. If the intermediate server receives SECTKNOVR with encrypted security-sensitive objects, then

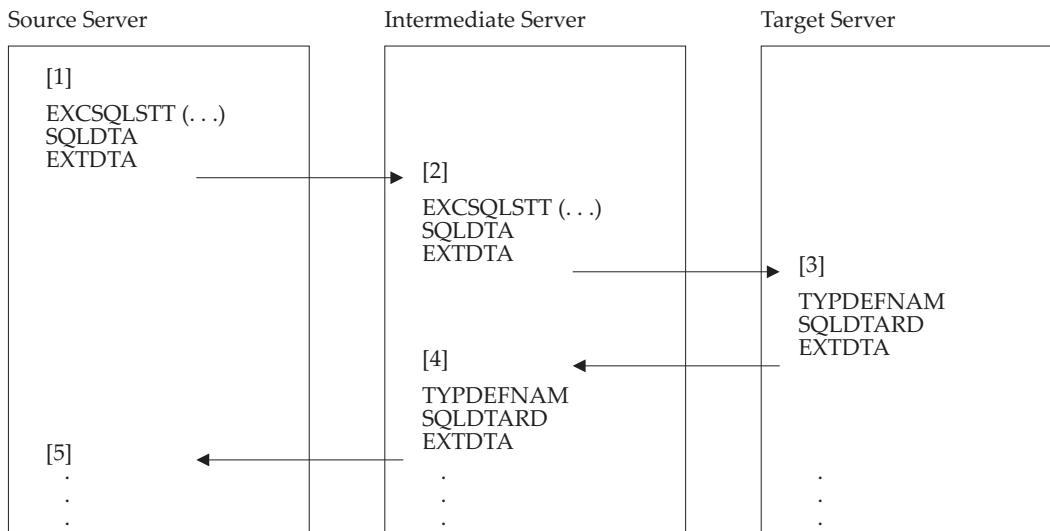
the intermediate server decrypts the encrypted SECTKNOVR. The intermediate server then re-encrypts the SECTKNOVR DSS.

3. The target server receives the encrypted SECTKNOVR and EXCSQLSTT with encrypted SQLDTA and encrypted EXTDTA. The target server first decrypts the SECTKNOVR. The target server then decrypts the encrypted security-sensitive objects using the decrypted encryption seed and encryption token in SECTKNOVR.

The target server generates a reply for the EXCSQLSTT command with encrypted SQLDTARD and encrypted EXTDTA. The target server returns the encrypted SQLDTARD and encrypted EXTDTA to the source server.

4. The intermediate server receives the encrypted SQLDTARD and encrypted EXTDTA. The intermediate server passes the two SECTKN objects used for encrypting the security-sensitive objects in SECTKNOVR and encrypts the SECTKNOVR DSS. If the intermediate server receives SECTKNOVR with encrypted security-sensitive objects, then the intermediate server decrypts the encrypted SECTKNOVR. The intermediate server then re-encrypts the SECTKNOVR DSS.
5. The source server receives encrypted SECTKNOVR, encrypted SQLDTARD, and encrypted EXTDTA. The source server first decrypts the SECTKNOVR. The source server then decrypts the encrypted security-sensitive objects using the decrypted encryption seed and encryption token in SECTKNOVR.

Figure 3-37 shows an example of the DDM commands and replies that flow when data is encrypted using the Encrypted User ID, Password, and Security-sensitive Data security mechanism that involves intermediate server processing and the upstream server does not support encryption.



**Figure 3-37** Intermediate Server when Upstream Site does not Support Encryption

The following is a brief description of some of the parameters for the DDM commands. See the appropriate terms for a detailed description of the parameters.

1. The source server generates an EXCSQLSTT command with encrypted SQLDTA and encrypted EXTDTA. The source passes the encrypted SQLDTA and encrypted EXTDTA with the EXCSQLSTT command.

2. The intermediate server receives the EXCSQLSTT, encrypted SQLDTA, and encrypted EXTDTA. If the upstream server does not support encryption, then the intermediate server decrypts the encrypted SQLDTA and EXTDTA, and sends the data in clear text.
3. The target server generates a reply for the EXCSQLSTT command with SQLDTARD and EXTDTA. The target server returns the SQLDTARD and EXTDTA to the source server.
4. The intermediate server receives the SQLDTARD and EXTDTA in clear text. The intermediate server encrypts the SQLDTARD and EXTDTA and sends the encrypted SQLDTARD and encrypted EXTDTA to the downstream server.
5. The source server receives encrypted SQLDTARD and encrypted EXTDTA. The source server decrypts the encrypted objects.

The details of the encryption and decryption processes are described in *DHENC* (on page 280). The details of the DSS format are described in *OBJDSS* (on page 536).

### Encrypted User ID and Security-Sensitive Data Security

In some environments, the only security token required is the user ID to ensure proper authorization for accessing data in the target system. One method to avoid sending the user ID and security-sensitive data in clear is to encrypt the user ID and security-sensitive data. The flows for this mechanism are identical to those for Encrypted User ID, Password, and Security-sensitive Data security, except that when the SECCHK command is sent to the server, only the encrypted values of the user ID flow in the SECTKN. The password is not sent to the server. The encrypted user ID is decrypted by the application server. The details of the encryption and decryption processes are described in *DHENC* (on page 280).

### Encrypted User ID, Password, New Password, and Security-Sensitive Data Security

One method to avoid sending the user ID, password, new password, and security-sensitive data in clear is to encrypt the user ID, password, new password, and security-sensitive data. The flows for this mechanism are identical to those for Encrypted User ID, Password, and Security-sensitive Data security, except that when the SECCHK command is sent to the server, the encrypted values of the user ID, password, and new password flow in three SECTKNs. The encrypted user ID, password, and new password are decrypted by the application server. The details of the encryption and decryption processes are described in *DHENC* (on page 280).

### SEE ALSO

<b>insvar</b>	<i>ACCSEC</i> (on page 52) <i>ACCSECRD</i> (on page 58)
<b>Semantic</b>	<i>DHENC</i> (on page 280) <i>ENCALG</i> (on page 331) <i>ENCKEYLEN</i> (on page 332) <i>EUSRIDDTA</i> (on page 355) <i>EUSRPWDDTA</i> (on page 361) <i>EXTDTA</i> (on page 397) <i>QRYDTA</i> (on page 686) <i>SECMEC</i> (on page 811) <i>SECMGR</i> (on page 814) <i>SECOVR</i> (on page 819) <i>SQLATTR</i> (on page 854) <i>SQLCINRD</i> (on page 857) <i>SQLDARD</i> (on page 859) <i>SQLDTA</i> (on page 860)

*SQLDTARD* (on page 862)  
*SQLRSLRD* (on page 867)  
*SQLSTT* (on page 868)  
*SQLSTTVRB* (on page 871)

**NAME**

ELMCLS — Element of Enumerated Class Attribute

**DESCRIPTION (Semantic)****Dictionary** QDDPRMD**Codepoint** X'004D'**Length** \***Class** CLASS**Sprcls** STRING - String

The Element of Enumerated Class Attribute (ELMCLS) String specifies that the elements of the array, index, or other collection being defined must contain elements of the specified class. The ELMCLS attribute, like the ENUCLS attribute, is used repeatedly to specify a list of possible classes for elements of the collection.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
<b>length</b>	*
<b>class</b>	X'004D'
<b>value</b>	INSTANCE_OF    CODPNTDR - Codepoint Data Representation REQUIRED
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO****insvar**            *DEFINITION* (on page 267)**Semantic**        *ARRAY* (on page 101)



**NAME**

ENCALG — Encryption Algorithm

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1909'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

The Encryption Algorithm (ENCALG) specifies the encryption standard algorithm to be used to encrypt and decrypt the security context information. ENCALG is used by the encryption security mechanisms. It may flow as an instance variable with ACCSEC and ACCSECRD. See *EDTASECOVR* (on page 323) for information on the DDM flows that carry the ENCALG.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'1909'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	16
	ENUVAL	1 - DES - Data Encryption Standard
	OPTIONAL	
	DFTVAL	1 - DES - Data Encryption Standard
	MINLVL	7
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** ACCSEC (on page 52)  
ACCSECRD (on page 58)

**Semantic** ACCSEC (on page 52)  
DHENC (on page 280)  
EDTASECOVR (on page 323)  
ENCKEYLEN (on page 332)  
EUSRIDDTA (on page 355)  
EUSRIDNWPWD (on page 357)  
EUSRIDPWD (on page 358)  
EUSRNPWDDTA (on page 359)  
EUSRPWDDTA (on page 361)

NAME

ENCKEYLEN — Encryption Key Length

DESCRIPTION (Semantic)

**Dictionary** QDDBASD

**Codepoint** X'190A'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

The Encryption Key Length (ENCKEYLEN) specifies the encryption key length to be used with ENCALG to encrypt and decrypt the security context information. ENCKEYLEN is used by the encryption security mechanisms. It may flow as an instance variable with ACCSEC and ACCSECRD. See *EDTASECOVR* (on page 323) for information on the DDM flows that carry the ENCKEYLEN.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'190A'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	16
	ENUVAL	1 - 56-bit encryption key
	OPTIONAL	
	DFTVAL	1 - 56-bit encryption key
	MINLVL	7
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

SEE ALSO

**insvar** ACCSEC (on page 52)  
ACCSECRD (on page 58)

**Semantic** ACCSEC (on page 52)  
DHENC (on page 280)  
EDTASECOVR (on page 323)  
EUSRIDDTA (on page 355)  
EUSRIDNWPWD (on page 357)  
EUSRIDPWD (on page 358)  
EUSRNPWDDTA (on page 359)  
EUSRPWDDTA (on page 361)

**NAME**

ENDATMCHN — End Atomic Chain

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD  
**Codepoint** X'1804'  
**Length** \*  
**Class** CODPNT  
**Sprcls** COMMAND - Command

The End Atomic Chain (ENDATMCHN) command marks the end of an atomic chain of Execute SQL Statement

**Source System Processing**

The ENDATMCHN command is used to terminate an atomic EXCSQLSTT chain that has previously been initiated with the Begin Atomic Chain (BGNATMCHN) command.

**DSS Carrier: RQSDSS**

**Target System Processing**

The *endchntyp* parameter specifies whether the atomic EXCSQLSTT chain is terminated normally or is aborted.

Normal completion results in an SQLCARD reply data object indicating success for the atomic chain. If the atomic EXCSQLSTT chain results in the first recoverable update to the RDB within the current unit of work, then an RDBUPDRM reply message must precede the SQLCARD.

**Exceptions**

Exception conditions the RDB detects are reported in an SQLCARD object. Other exception conditions are indicated by an error reply message.

**Source System Reply Processing**

Return any error reply messages to the requester.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'1804'	
endchntyp	INSTANCE_OF	ENDCHNTYP - End Atomic Chain Type
	OPTIONAL	
	MINLVL	7
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

<b>cmddta</b>	<b>NIL</b>	
<b>rpydta</b>		<b>REPLY OBJECTS</b>
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL ENUVAL DFTVAL NOTE	TYPDEFNAM - Data Type Definition Name 'QTDSQL370' 'QTDSQL400' 'QTDSQLX86' 'QTDSQLASC' 'QTDSQLVAX' The default means the value received on ACCRDBRM is used.
	OPTIONAL	
X'0035'	INSTANCE_OF DFTVAL NOTE	TYPDEFOVR - TYPDEF Overrides The default means the value received on ACCRDBRM is used.
	OPTIONAL	
X'002F'	INSTANCE_OF REQUIRED NOTE	SQLCARD - SQL Communications Area Reply Data The SQLCARD is required if the chain is processed successfully, or if the ENDATMCHN results in an error detected by the RDB.
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported
X'1245'	INSTANCE_OF	PRCCNVRM - Protocol Conversation Error
X'2218'	INSTANCE_OF	RDBUPDRM - RDB Update Reply Message
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limit Reached
X'2213'	INSTANCE_OF	SQLERRRM - SQL Error Condition
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

**SEE ALSO**

<b>insvar</b>	<i>ENDATMCHN</i> (on page 333) <i>ENDCHNTYP</i> (on page 341) <i>EXCSQLSTT</i> (on page 381) <i>MINLVL</i> (on page 518)
<b>rpydta</b>	<i>ACCRDBRM</i> (on page 48) <i>AGNPRMRM</i> (on page 65) <i>CMDCHKRM</i> (on page 173) <i>CMDNSPRM</i> (on page 176) <i>PRCCNVRM</i> (on page 625) <i>RDBUPDRM</i> (on page 751) <i>RSCLMTRM</i> (on page 778) <i>SQLCARD</i> (on page 855) <i>SQLERRRM</i> (on page 864) <i>SYNTAXRM</i> (on page 989)

**Semantic**

*TYPDEFNAM* (on page 1027)

*TYPDEFQVR* (on page 1030)

*VALNSPRM* (on page 1057)

*BGNATMCHN* (on page 107)

*CODPNT* (on page 233)

*COMMAND* (on page 240)

*EXCSQLSTT* (on page 381)

*QDDBASD* (on page 651)

*RDBUPDRM* (on page 751)

*RQSDSS* (on page 774)

*SQLCARD* (on page 855)

## NAME

ENDBND — End Binding a Package to an RDB

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD

**Codepoint** X'2009'

**Length** \*

**Class** CLASS

**Sprcls** COMMAND - Command

The End Binding a Package to an RDB (ENDBND) Command terminates the process of binding a package to a relational database (RDB).

**Source System Processing**

The source system determines the location of the RDB:

Local Call the local RDB server.

Remote Send the ENDBND command to the remote RDB server.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**DSS Carrier: RQSDSS****Target System Processing**

The *maxsctnbr* parameter specifies the maximum number of sections the source server program preparation process (the precompiler) assigns for this package. The target RDB must ensure that the package contains the specified number of sections. To meet this requirement, the target RDB might add one or more sections that are reserved for the dynamic Structured Query Language (SQL) to the package before terminating the bind process.

The *pkgnamct* parameter specifies the fully qualified package name and its consistency token. The *pkgnamct* parameter must be the same as the *pkgnamct* parameter specified on the BGNBND command that started the package binding process.

The *rdbnam* parameter specifies the name of the RDB to which the ACCRDB command gained access. If the *rdbnam* parameter is specified, its value must be the same as the *rdbnam* parameter on the ACCRDB command.

The ENDBND command might cause a package to be created or replaced depending on the *bndcrtctl* parameter value specified on the BGNBND command and on the errors detected during the bind process. See Figure 3-38 (on page 337) for an illustration of a package the ENDBND command creates. If a package is created, it contains (or has associated with it) at least:

- A copy of all the BGNBND parameters and the defaults for any unspecified optional BGNBND parameters
- Sections for all SQL statements bound by BNDSQLSTT commands executed between the BGNBND command and this ENDBND command

- The SQL statement text and variable definitions for each SQL statement bound as stated above
- Reserved sections for SQL statements that did not flow at bind time

If the package contains reserved sections (because of the *maxsctnbr* parameter or because the *BNDSQLSTT* command skipped section numbers), the RDB must treat each ambiguous database cursor (see *QRYBLKCTL* (on page 672)) in the package as a potential target of a *WHERE\_CURRENT\_OF* clause on an SQL UPDATE or DELETE statement. This can affect the choice of query protocols that the target SQLAM uses. See *OPNQRY* (on page 555) and *QRYBLKCTL* (on page 672) for more information about query protocols.

Assume the following command sequence has occurred:

```
BGNBND ( RDBNAM (RDBxxxxxxxxxxxxxxxx)
         PKGNAMCT (RDBxxx, collid, package1, 12345678)
         VRSNAM (version1) BNDCRTCTL (BNDERRALW)
         BNDCHKEXS (BNDEXSOPT) PKGRPLOPT (PKGRPLALW)
         PKGATHOPT (PKGATHRVK)
         STTSTRDEL (STRDELAP) STTDECDEL (DECDELPRD)
         STTDATFMT (USADATFMT) STTTIMFMT (USATIMFMT)
         PKGDFTCST (CSTBITS) RDBRLSOPT (RDBRLSCMM)
         BNDEXPOPT (EXPNON) PKGOWNID (BOB)
         QRYBLKCTL (FIXROWPRC) DECPRC (15)
         PKGDFTC (CCSIDSBC (01F4) CCSIDDBC (012D) ) )

BNDSQLSTT ( RDBNAM (RDBxxxxxxxxxxxxxxxx)
            PKGNAMCSN (RDBxxx, collid, package1, 12345678, 03)
            SQLSTTNBR (5) )
SQLSTT ( 'DECLARE CURSOR ...' )
SQLSTTVRB (... )

BNDSQLSTT ( RDBNAM (RDBxxxxxxxxxxxxxxxx)
            PKGNAMCSN (RDBxxx, collid, package1, 12345678, 05)
            SQLSTTNBR (7) )
SQLSTT ( 'UPDATE ...' )
SQLSTTVRB (... )

ENDBND ( RDBNAM (RDBxxxxxxxxxxxxxxxx)
         PKGNAMCT (RDBxxx, collid, package1, 12345678)
         MAXSCTNBR (8) )
```

**Figure 3-38** An Example of Package Creation by ENDBND

For this sequence of commands, the package created would contain:

1. The BGNBND command parameter values
2. Eight sections:
  - Sections 1, 2, 4, 6, 7, and 8 are reserved sections for SQL statements that did not flow at bind time.
  - Sections 3 and 5 contain the SQL statements sent with the BNDSQLSTT commands.
3. The SQLSTT and SQLSTTVRB object contents for sections 3 and 5

Normal completion of the ENDBND command returns an SQLCARD object. An RDBUPDRM must be returned if the command makes recoverable updates in the RDB and if a server has not

returned a prior RDBUPDRM in the current unit of work. A recoverable update is an RDB update that writes information to the recovery log of the RDB.

The TYPDEFNAM reply data object specifies the name of the data type to data representation mapping definitions that the target SQLAM uses when sending the SQLCARD object for this command.

The TYPDEFOVR reply data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type to data representation mapping definitions for the SQLCARD object for this command.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**Exceptions**

Exception conditions the RDB detects are reported in an SQLCARD object.

The *bndcrctl* parameter value, specified on the BGNBND command, determines whether the package is created.

If the package binding process is not active, then the command is rejected with the PKGBNARM.

If access to the specified RDB is not current, then the command is rejected with the RDBNACRM.

**Source System Reply Processing**

Return any reply messages and data objects to the requester.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
<b>length</b>	<b>*</b>	
<b>class</b>	<b>X'2009'</b>	
<b>maxsctnbr</b>	<b>INSTANCE_OF OPTIONAL</b>	<b>MAXSCTNBR - Maximum Section Number</b>
<b>pkgnamct</b>	<b>INSTANCE_OF REQUIRED</b>	<b>PKGNAMECT - RDB Package Name and Consistency Token</b>
<b>rdbnam</b>	<b>INSTANCE_OF OPTIONAL CMDTRG</b>	<b>RDBNAM - Relational Database Name</b>
<b>clscmd</b>	<b>NIL</b>	



<b>inscmd</b>	<b>NIL</b>	
<b>cmddta</b>	<b>NIL</b>	
<b>rpydta</b>		<b>REPLY OBJECTS</b>
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL MINLVL ENUVAL MINLVL OPTIONAL DFTVAL NOTE	TYPDEFNAM - Data Type Definition Name 'QTDSQL370' 'QTDSQL400' 'QTDSQLX86' 'QTDSQLASC' 4 'QTDSQLVAX' 4 " The default means the value received on the ACCRDBRM is used.
X'0035'	INSTANCE_OF OPTIONAL DFTVAL NOTE	TYPDEFOVR - TYPDEF Overrides " The default means the value received on the ACCRDBRM is used.
X'1C03'	INSTANCE_OF OPTIONAL MINLVL	MGRVLVLOVR - Manager Level Overrides 7
X'2408'	INSTANCE_OF REQUIRED	SQLCARD - SQL Communications Area Reply Data
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'220D'	INSTANCE_OF	ABNUOWRM - Abnormal End Unit of Work Condition
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported
X'2206'	INSTANCE_OF	PKGBNARM - RDB Package Binding Not Active
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'2204'	INSTANCE_OF	RDBNACRM - RDB Not Accessed
X'2218'	INSTANCE_OF MINLVL	RDBUPDRM - RDB Update Reply Message 4
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limits Reached
X'2213'	INSTANCE_OF	SQLERRRM - SQL Error Condition
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'125F'	INSTANCE_OF	TRGNSPRM - Target Not Supported
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

**SEE ALSO****Semantic**

*BGNBND* (on page 110)  
*BNDSQLSTT* (on page 136)  
*PKGBNARM* (on page 585)  
*PKGBPARM* (on page 586)  
*PKGOWNID* (on page 600)  
*REBIND* (on page 753)  
*SQL* (on page 835)  
*SQLAM* (on page 847)

**NAME**

ENDCHNTYP — End Atomic Chain Type

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1902'

**Length** \*

**Class** CLASS

**Sprcls** SCALAR - Scalar Object

End Chain Type (ENDCHNTYP) specifies for an atomic chain of Execute SQL Statement (EXCSQLSTT) commands whether the chain is to be terminated normally or is to be aborted.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	5	
class	X'1902'	
value	INSTANCE_OF	BYTDR - An 8-bit Data Representation Value
	ENUVAL	X'00' - Atomic EXCSQLSTT chain is terminated normally.
	ENUVAL	X'01' - Atomic EXCSQLSTT chain is aborted.
	OPTIONAL	
	DFTVAL	X'00' - Atomic EXCSQLSTT chain is terminated normally.
	MINLVL	7
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *BYTDR* (on page 145)  
*ENDATMCHN* (on page 333)  
*EXCSQLSTT* (on page 381)  
*MINLVL* (on page 518)

**Semantic** *CLASS* (on page 158)  
*EXCSQLSTT* (on page 381)  
*QDDBASD* (on page 651)  
*SCALAR* (on page 796)

**NAME**

ENDQRYRM — End of Query

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'220B'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

The End of Query (ENDQRYRM) Reply Message indicates that the query process has terminated in such a manner that the query or result set is now closed. It cannot be resumed with the CNTQRY command or closed with the CLSQRY command.

The ENDQRYRM is always followed by an SQLCARD object.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'220B'	
<b>rdbnam</b>	INSTANCE_OF OPTIONAL	RDBNAM - Relational Database Name
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED	SVRCOD - Severity Code
	ENUVAL	4 - WARNING - Warning Severity Code
	ENUVAL	8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**cmdrpy** CNTQRY (on page 222)  
EXCSQLSTT (on page 381)  
OPNQRY (on page 555)

**Semantic** CLSQRY (on page 165)  
CNTQRY (on page 222)  
FIXROWPRC (on page 417)  
LMTBLKPRC (on page 475)  
OPNQRY (on page 555)  
QRYBLK (on page 670)  
QRYNOPRM (on page 690)  
SQLAM (on page 847)

**NAME**

ENDUOWRM — End Unit of Work Condition

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'220C'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

The End Unit of Work Condition (ENDUOWRM) Reply Message specifies that the unit of work has ended as a result of the last command.

**Source System Processing**

If the connection is to be reused at the completion of the commit or rollback, the release connection parameter (*rlsconv*) is set to REUSE. The connection cannot be reused until the receipt of the ENDUOWRM. If the target server responds with *rlsconv* set to FALSE, the connection is not allowed to be reused until the application terminates. If set to REUSE, the target server returns a set of SQLSTT objects to establish the application execution environment.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'220E'	
<b>rdbnam</b>	INSTANCE_OF OPTIONAL	RDBNAM - Relational Database Name
<b>rlsconv</b>	INSTANCE_OF ENUVAL ENUVAL NOTE  NOTE  DFTVAL OPTIONAL MINLVL	RLSCONV - Release Connection X'F0'(NO) - Do not reuse connection X'F2' (REUSE) - Connection can be reused This parameter is required if the RLSCONV parameter on an RDBCMM specified REUSE. Otherwise, this parameter can not be specified. X'F1' (TERMINATE) is not allowed. If specified, it is a conversation protocol error. X'F0' (NO) - Do not reuse connection 7
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code  4 - WARNING - Warning Severity Code
<b>uowdsp</b>	INSTANCE_OF REQUIRED	UOWDSP - Unit of Work Disposition

<b>rpydta</b>	SQLSTT
<b>srvdgn</b>	INSTANCE_OF SRVDGN - Server Diagnostic Information OPTIONAL
<b>clscmd</b>	NIL
<b>inscmd</b>	NIL

**SEE ALSO**

<b>cmdrpy</b>	<i>EXCSQLIMM</i> (on page 371) <i>EXCSQLSET</i> (on page 377) <i>EXCSQLSTT</i> (on page 381) <i>RDBCMM</i> (on page 728) <i>RDBRLLBCK</i> (on page 745)
<b>Semantic</b>	<i>EXCSQLIMM</i> (on page 371) <i>EXCSQLSTT</i> (on page 381) <i>QRYNOPRM</i> (on page 690) <i>RDBCMM</i> (on page 728) <i>RDBRLLBCK</i> (on page 745)

**NAME**

ENUCLS — Enumerated Class Attribute

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'0040'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

The Enumerated Class Attribute (ENUCLS) String specifies that the attribute value is the instance of the class which is specified for a parameter or value.

The ENUCLS attribute is repeated in the description of a parameter or value to specify the complete list of valid classes.

Only one of the enumerated classes is specified unless the REPEATABLE attribute is also specified. In that case, any number of values can be specified, but each of them must be of one of the enumerated classes.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
<b>length</b>	*
<b>class</b>	X'0040'
<b>value</b>	INSTANCE_OF    CODPNTDR - Codepoint Data Representation REQUIRED
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO**

- insvar**            *DCTINDEN* (on page 259)  
                      *TRGNSPRM* (on page 1022)
- Semantic**        *ELMCLS* (on page 330)

**NAME**

ENULEN — Enumerated Length Attribute

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'0015'

**Length** \*

**Class** CLASS

**Sprcls** BIN - Binary Integer Number

The Enumerated Length Attribute (ENULEN) specifies that the attribute value is one of the valid lengths for the term.

The ENULEN attribute is repeated in the description of a term to specify the complete list of valid lengths.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'0015'	
<b>value</b>	INSTANCE_OF	BINDR - Binary Number Field
	ENULEN	16
	ENULEN	32
	ENULEN	48
	ENULEN	64
	NOTE	Other valid enumerated lengths are multiples of 2 bytes (48, 64, 80, and so on).
	REQUIRED	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *BINDR* (on page 120)  
*LENGTH* (on page 472)  
*MAXLEN* (on page 495)  
*MINLEN* (on page 517)



**NAME**

ENUVAL — Enumerated Value Attribute

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD  
**Codepoint** X'0016'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

The Enumerated Value Attribute (ENUVAL) specifies that the attribute value is one of the values that is specified for a term.

The ENUVAL attribute is repeated in the description of a term to specify the complete list of valid values.

The attribute value must have attributes compatible with those of the term being described.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
<b>length</b>	*	
<b>class</b>	X'0016'	
<b>value</b>	SPRCLS NOTE	FIELD - A Discrete Unit of Data The attributes for this variable vary according to what is being defined. Also, when BITDR, BITSTRDR, HEXDR, or HEXSTRDR values are specified, they are represented as being left-justified in the fewest number of whole bytes possible. For example, 123 would be X'1230'.
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**clsvar** QRYPRCTYP (on page 692)  
**cmddda** BGNBND (on page 110)  
 BNDSQLSTT (on page 136)  
 DSCRDBTBL (on page 300)  
 EXCSQLIMM (on page 371)  
 EXCSQLSTT (on page 381)  
 OPNQRY (on page 555)  
 PRPSQLSTT (on page 636)  
**insvar** ABNUOWRM (on page 39)  
 ACCRDB (on page 42)  
 ACCRDBRM (on page 48)  
 AGNPRMRM (on page 65)  
 BGNBNDRM (on page 117)  
 BITDR (on page 123)  
 BNDCHKEXS (on page 125)  
 BNDCRTCTL (on page 127)

*BNDXPOPT* (on page 129)  
*BNDSTTASM* (on page 141)  
*BOOLEAN* (on page 142)  
*CMDATHRM* (on page 171)  
*CMDCHKRM* (on page 173)  
*CMDCMPRM* (on page 175)  
*CMDNSPRM* (on page 176)  
*CMDVLTRM* (on page 181)  
*CMMRQSRM* (on page 182)  
*CMMTYP* (on page 183)  
*DECPRC* (on page 266)  
*DEPERRC* (on page 270)  
*DSCERRCD* (on page 296)  
*DSCINVRM* (on page 298)  
*DSSFMT* (on page 318)  
*DTAMCHRM* (on page 320)  
*ENDQRYRM* (on page 342)  
*ENDUOWRM* (on page 343)  
*FORGET* (on page 423)  
*MGRDEPRM* (on page 505)  
*MGRLVL* (on page 506)  
*MGRLVLRM* (on page 512)  
*NAMSYMDR* (on page 528)  
*OBJDSS* (on page 536)  
*OBJNSPRM* (on page 542)  
*OPNQFLRM* (on page 554)  
*OPNQRY* (on page 555)  
*OPNQRYRM* (on page 566)  
*OUTEXP* (on page 571)  
*OUTOVROPT* (on page 576)  
*PKGATHOPT* (on page 580)  
*PKGATHRUL* (on page 581)  
*PKGBNARM* (on page 585)  
*PKGBPARM* (on page 586)  
*PKGDFTCST* (on page 590)  
*PKGISOLVL* (on page 592)  
*PKGRPLOPT* (on page 604)  
*PRCCNVCD* (on page 621)  
*PRMNSPRM* (on page 633)  
*QRYBLKCTL* (on page 672)  
*QRYNOPRM* (on page 690)  
*QRYPOPRM* (on page 691)  
*RDBACCCL* (on page 723)  
*RDBACCRM* (on page 724)  
*RDBAFLRM* (on page 725)  
*RDBALWUPD* (on page 726)  
*RDBATHRM* (on page 727)  
*RDBCMTOK* (on page 731)  
*RDBNACRM* (on page 734)  
*RDBNFNRM* (on page 739)  
*RDBRLSOPT* (on page 750)  
*RDBUPDRM* (on page 751)

	<i>RLSCONV</i> (on page 765)
	<i>RPYDSS</i> (on page 766)
	<i>RQSDSS</i> (on page 774)
	<i>RSCLMTRM</i> (on page 778)
	<i>RSCTYP</i> (on page 782)
	<i>RSLSETFLG</i> (on page 783)
	<i>RSLSETRM</i> (on page 785)
	<i>RSYNCTYP</i> (on page 790)
	<i>RTNEXTDTA</i> (on page 792)
	<i>RTNSQLDA</i> (on page 795)
	<i>SECCHKCD</i> (on page 806)
	<i>SECCHKRM</i> (on page 809)
	<i>SECMEC</i> (on page 811)
	<i>SPVNAM</i> (on page 834)
	<i>SQLCSRHLD</i> (on page 858)
	<i>SQLERRRM</i> (on page 864)
	<i>STTDATFMT</i> (on page 894)
	<i>STTDECDEL</i> (on page 897)
	<i>STTSTRDEL</i> (on page 899)
	<i>STTTIMFMT</i> (on page 900)
	<i>SVRCOD</i> (on page 911)
	<i>SYNCCTL</i> (on page 915)
	<i>SYNCTYPE</i> (on page 984)
	<i>SYNERRCD</i> (on page 985)
	<i>SYNTAXRM</i> (on page 989)
	<i>TRGNSPRM</i> (on page 1022)
	<i>TYPDEFNAM</i> (on page 1027)
	<i>TYPFMLNM</i> (on page 1033)
	<i>TYPSQLDA</i> (on page 1034)
	<i>UOWDSP</i> (on page 1037)
	<i>UOWSTATE</i> (on page 1040)
	<i>VALNSPRM</i> (on page 1057)
<b>rpydta</b>	<i>ACCRDB</i> (on page 42)
	<i>BNDSQLSTT</i> (on page 136)
	<i>CLSQRY</i> (on page 165)
	<i>CNTQRY</i> (on page 222)
	<i>DRPPKG</i> (on page 293)
	<i>DSCRDBTBL</i> (on page 300)
	<i>DSCSQLSTT</i> (on page 304)
	<i>ENDBND</i> (on page 336)
	<i>EXCSQLIMM</i> (on page 371)
	<i>EXCSQLSTT</i> (on page 381)
	<i>OPNQRY</i> (on page 555)
	<i>PRPSQLSTT</i> (on page 636)
	<i>RDBCMM</i> (on page 728)
	<i>RDBRLLBCK</i> (on page 745)
	<i>REBIND</i> (on page 753)
<b>Semantic</b>	<i>LVLCMP</i> (on page 486)
	<i>QLFATT</i> (on page 664)
	<i>SUBSETS</i> (on page 902)

**NAME**

ERROR — Error Severity Code

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'0017'

**Length** \*

**Class** CONSTANT

The Error Severity Code (ERROR) specifies that an error condition was detected in the processing of a command. All effects of the condition have been backed out or prevented. For example, any effects on cursor positioning or locks obtained or released have been backed out. Conditions must be the same as before the command was issued. For example, if a record is unlocked, obtaining the record lock again is not sufficient; the record contents must not have changed while the record was unlocked.

Further processing of a command depends on the architected specifications of the specific command, the error condition, and the environment in which it is being executed. For example, a File Not Found (FILNFNRM) error always causes a command to be terminated, but a Duplicate File (DUPFILRM) error terminates processing of a CRTxxxF command only if that is specified by the Duplicate File Option (DUPFILOP) parameter.

---

value 8

**SEE ALSO**

- insvar**
- ACCSECRD* (on page 58)
  - BGNBNDRM* (on page 117)
  - CMDATHRM* (on page 171)
  - CMDCHKRM* (on page 173)
  - CMDNSPRM* (on page 176)
  - CMDVLTRM* (on page 181)
  - CMMRQSRM* (on page 182)
  - DSCINVRM* (on page 298)
  - DTAMCHRM* (on page 320)
  - ENDQRYRM* (on page 342)
  - MGRDEPRM* (on page 505)
  - MGRLVLRM* (on page 512)
  - OBJNSPRM* (on page 542)
  - OPNQFLRM* (on page 554)
  - PKGBNARM* (on page 585)
  - PKGBPARM* (on page 586)
  - PRCCNVRM* (on page 625)
  - PRMNSPRM* (on page 633)
  - QRYNOPRM* (on page 690)
  - QRYPOPRM* (on page 691)
  - RDBACCRM* (on page 724)
  - RDBAFLRM* (on page 725)
  - RDBATHRM* (on page 727)
  - RDBNACRM* (on page 734)
  - RDBNFNRM* (on page 739)
  - RSCLMTRM* (on page 778)

**Semantic**

*SECCHKRM* (on page 809)  
*SQLERRRM* (on page 864)  
*SVRCOD* (on page 911)  
*SYNTAXRM* (on page 989)  
*TRGNSPRM* (on page 1022)  
*VALNSPRM* (on page 1057)

*ABNUOWRM* (on page 39)  
*APPSRCER* (on page 91)  
*APPTRGER* (on page 95)  
*BNDNERALW* (on page 132)  
*CMDCHKRM* (on page 173)  
*DSS* (on page 308)  
*SECCHKCD* (on page 806)  
*TCPTRGER* (on page 1015)

**NAME**

ETIME — Elapsed Time

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1901'

**Length** \*

**Class** CLASS

**Sprcls** FIELD - A Discrete Unit of Data

The elapsed time is a 64-bit binary number which measures time in microseconds.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>INSTANCE VARIABLES</b>	
length	12	
class	X'1901	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH_OF	64
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

None.

**NAME**

EURDATFMT — European Date Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'242B'

**Length** \*

**Class** CODPNT

The European Date Format (EURDATFMT) specifies that dates in the Structured Query Language (SQL) statements are in the European date format:

dd.mm.yyyy

where "." is a period character (with the Graphic Character Global Identifier (GCGID) SP11). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar** *STTDATFMT* (on page 894)

**NAME**

EURTIMFMT — European Time Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2430'

**Length** \*

**Class** CODPNT

The European Time Format (EURTIMFMT) specifies that times in the Structured Query Language (SQL) statements are in the European time format:

hh.mm.ss

where "." is a period character (with the Graphic Character Global Identifier (GCGID) SP11). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar** *STTIMFMT* (on page 900)

**Semantic** *HMSPRDTIMFMT* (on page 434)  
*STTIMFMT* (on page 900)



**NAME**

EUSRIDDTA — Encrypted User ID and Encrypted Security-sensitive Data

**DESCRIPTION (Semantic)****Dictionary** QDDPRMD**Length** \***Class** CONSTANT

The Encrypted User ID and Encrypted Security-sensitive Data Mechanism (EUSRIDDTA) specifies the use of encrypted user ID and security-sensitive data mechanism. A shared private key is generated by exchanging connection keys in the SECTKN instance variable on the ACCSEC command and the ACCSECRD reply data using the standard Diffie-Hellman key distribution algorithm. The encryption algorithm and the encryption key length are exchanged on the ACCSEC command and the ACCSECRD reply data by passing the encryption algorithm in the ENCALG instance variable, and the encryption key length in the ENCKEYLEN instance variable. The encryption seed is generated from the Diffie-Hellman shared private key as specified by the combination of encryption algorithm and encryption key length string. The encryption token is generated from the server connection key as specified by the combination of encryption algorithm and encryption key length.

The application requester encrypts the user ID using the encryption seed and the encryption token. The encrypted user ID is sent in the SECTKN instance variable on the SECCHK command. The application server decrypts the user ID using the encryption seed and the encryption token. The plain text user ID is then passed to the local security server to be authenticated.

On successful connection, if the request data stream contains security-sensitive objects, then the application requester encrypts the DSS carrier for the security-sensitive objects using the encryption token and the encryption seed generated during connect processing. The application requester sets the *dsstype*, Encrypted OBJDSS to indicate that the object is encrypted. The application server decrypts the encrypted objects using the encryption token and the encryption seed generated during connect processing.

If the reply data stream contains security-sensitive objects, then the application server encrypts the DSS carrier for the security-sensitive objects using the encryption token and the encryption seed. The application server sets the *dsstype*, Encrypted OBJDSS to indicate that the object is encrypted. The application requester decrypts the encrypted objects using the encryption token and the encryption seed.

See the following terms for the specific meaning and function of the security mechanism:

DHENC            Diffie-Hellman Encryption Security Mechanism (see *DHENC* (on page 280))

EDTASECOVR    Encrypted User ID Password and Data Security Overview (see *EDTASECOVR* (on page 323))

EUSRNPWDDTA Encrypted User ID, Encrypted Password, Encrypted New Password, and Encrypted Security-sensitive Data (see *EUSRNPWDDTA* (on page 359))

EUSRPWDDTA   Encrypted User ID Encrypted Password and Encrypted Security-sensitive Data (see *EUSRPWDDTA* (on page 361))

SECTKNOVR     Security Token Override (see *SECTKNOVR* (on page 822))

---

value	12
-------	----

**SEE ALSO**

<b>insvar</b>	<i>ACCSEC</i> (on page 52) <i>ACCSECRD</i> (on page 58) <i>SECCHK</i> (on page 800)
<b>Semantic</b>	<i>DHENC</i> (on page 280) <i>EUSRPWDDTA</i> (on page 361) <i>SECCHK</i> (on page 800)

**NAME**

EUSRIDNWPWD — Encrypted User ID, Encrypted Password, and Encrypted New Password

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD  
**Length** \*  
**Class** CONSTANT

The Encrypted User ID, Encrypted Password, and Encrypted New Password Mechanism (EUSRIDNWPWD) specifies the use of the encrypted user ID, password, and new password mechanism. The mechanism is similar to the New Password security mechanism, but the user ID, password, and new password are encrypted. A shared private key is generated by exchanging connection keys in the SECTKN instance variable on the ACCSEC command and the ACCSECRD reply data, using the standard Diffie-Hellman key distribution algorithm. The encryption algorithm and the encryption key length are exchanged on the ACCSEC command and the ACCSECRD reply data by passing the encryption algorithm in the ENCALG instance variable, and the encryption key length in the ENCKEYLEN instance variable. The encryption seed is generated from the Diffie-Hellman shared private key as specified by the combination of encryption algorithm and encryption key length. The encryption token is generated from the server connection key as specified by the combination of encryption algorithm and encryption key length.

The application requester encrypts the user ID, password, and new password using the encryption seed and the encryption token. The encrypted user ID, password, and new password are sent in three SECTKN instance variables on the SECCHK command, user ID SECTKN first, password SECTKN second, new password SECTKN third. The application server decrypts the user ID, password, and new password using the encryption seed and the encryption token. The plain text user ID, password, and new password are then passed to the local security server to be authenticated.

---

value	10
-------	----

**SEE ALSO**

**insvar**      *ACCSEC* (on page 52)  
                  *ACCSECRD* (on page 58)  
                  *SECCHK* (on page 800)  
                  *SECMEC* (on page 811)

**Semantic**    *DHENC* (on page 280)  
                  *SECCHK* (on page 800)

**NAME**

EUSRIDPWD — Encrypted User ID and Encrypted Password

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD  
**Length** \*  
**Class** CONSTANT

The Encrypted User ID and Encrypted Password Mechanism (EUSRIDPWD) specifies the use of the encrypted user ID and password mechanism. The mechanism authenticates the user like the user ID and password mechanism, but the user ID and password are encrypted. A shared private key is generated by exchanging connection keys in the SECTKN instance variable on the ACCSEC command and the ACCSECRD reply data, using the standard Diffie-Hellman key distribution algorithm. The encryption algorithm and the encryption key length are exchanged on the ACCSEC command and the ACCSECRD reply data by passing the encryption algorithm in the ENCALG instance variable, and the encryption key length string in the ENCKEYLEN instance variable. The encryption seed is generated from the Diffie-Hellman shared private key, as specified by the combination of encryption algorithm and encryption key length. The encryption token is generated from the server connection key as specified by the combination of encryption algorithm and encryption key length.

The application requester encrypts the user ID and the password using the encryption seed and the encryption token. The encrypted user ID and password are sent in two SECTKN instance variables on the SECCHK command, user ID SECTKN first, password SECTKN second. The application server decrypts the user ID and password using the encryption seed and the encryption token. The plain text user ID and password are then passed to the local security server to be authenticated.

See the following terms for the specific meaning and function of each security mechanism component:

DHENC	Diffie-Hellman Encryption Security Mechanism (see <i>DHENC</i> (on page 280))
EUSRIDNWPWD	Encrypted User ID New Password Security Mechanism (see <i>EUSRIDNWPWD</i> (on page 357))
NWPWDSEC	New Password Security Mechanism (see <i>NWPWDSEC</i> (on page 535))
USRIDSEC	User Identification Security Mechanism (see <i>USRIDSEC</i> (on page 1048))
USRSECOVR	User ID Security Overview (see <i>USRSECOVR</i> (on page 1050))

---

value	9
-------	---

**SEE ALSO**

<b>insvar</b>	<i>ACCSEC</i> (on page 52) <i>ACCSECRD</i> (on page 58) <i>SECMEC</i> (on page 811)
<b>Semantic</b>	<i>DHENC</i> (on page 280) <i>SECCHK</i> (on page 800)

**NAME**

EUSRNPWDDTA — Encrypted User ID, Encrypted Password, Encrypted New Password, and Encrypted Security-sensitive Data

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Length** \*

**Class** CONSTANT

The Encrypted User ID, Encrypted Password, Encrypted New Password, and Encrypted Security-sensitive Data Mechanism (EUSRNPWDDTA) specifies the use of the encrypted user ID, password, new password, and security-sensitive data mechanism. A shared private key is generated by exchanging connection keys in the SECTKN instance variable on the ACCSEC command and the ACCSECRD reply data, using the standard Diffie-Hellman key distribution algorithm. The encryption algorithm and the encryption key length are exchanged on the ACCSEC command and the ACCSECRD reply data by passing the encryption algorithm in the ENCALG instance variable, and the encryption key length in the ENCKEYLEN instance variable. The encryption seed is generated from the Diffie-Hellman shared private key as specified by the combination of encryption algorithm and encryption key length. The encryption token is generated from the server connection key as specified by the combination of encryption algorithm and encryption key length.

The application requester encrypts the user ID, password, and new password using the encryption seed and the encryption token. The encrypted user ID, password, and new password are sent in three SECTKN instance variables on the SECCHK command, user ID SECTKN first, password SECTKN second, and new password SECTKN third. The application server decrypts the user ID, password, and new password using the encryption seed and the encryption token. The plain text user ID, password, and new password are then passed to the local security server to be authenticated.

On successful connection, if the request data stream contains security-sensitive objects, then the application requester encrypts the DSS carrier for the security-sensitive objects using the encryption token and the encryption seed generated during connect processing. The application requester sets the *dsstype*, Encrypted OBJDSS to indicate that the object is encrypted. The application server decrypts the encrypted objects using the encryption token and the encryption seed generated during connect processing.

If the reply data stream contains security-sensitive objects, then the application server encrypts the DSS carrier for the security-sensitive objects using the encryption token and the encryption seed. The application server sets the *dsstype*, Encrypted OBJDSS to indicate that the object is encrypted. The application requester decrypts the encrypted objects using the encryption token and the encryption seed.

See the following terms for the specific meaning and function of the security mechanism:

DHENC	Diffie-Hellman Encryption Security Mechanism (see <i>DHENC</i> (on page 280))
EDTASECOVR	Encrypted User ID Password and Data Security Overview (see <i>EDTASECOVR</i> (on page 323))
EUSRIDDTA	Encrypted User ID and Encrypted Security-sensitive Data (see <i>EUSRIDDTA</i> (on page 355))
EUSRPWDDTA	Encrypted User ID, Encrypted Password, and Encrypted Security-sensitive Data (see <i>EUSRPWDDTA</i> (on page 361))

SECTKNOVR Security Token Override (see *SECTKNOVR* (on page 822))

---

value 13

**SEE ALSO**

**insvar**      *ACCSEC* (on page 52)  
                 *ACCSECRD* (on page 58)  
                 *SECCHK* (on page 800)  
                 *SECMEC* (on page 811)

**Semantic**    *DHENC* (on page 280)  
                 *EUSRIDDTA* (on page 355)  
                 *EUSRPWDDTA* (on page 361)  
                 *SECMEC* (on page 811)

**NAME**

EUSRPWDDTA — Encrypted User ID, Encrypted Password, and Encrypted Security-sensitive Data

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD  
**Length** \*  
**Class** CONSTANT

The Encrypted User ID, Encrypted Password, and Encrypted Security-sensitive Data Mechanism (EUSRPWDDTA) specifies the use of the encrypted user ID, password, and security-sensitive data mechanism. A shared private key is generated by exchanging connection keys in the SECTKN instance variable on the ACCSEC command and the ACCSECRD reply data, using the standard Diffie-Hellman key distribution algorithm. The encryption algorithm and the encryption key length are exchanged on the ACCSEC command and the ACCSECRD reply data by passing the encryption algorithm in the ENCALG instance variable and the encryption key length in the ENCKEYLEN instance variable. The encryption seed is generated from the Diffie-Hellman shared private key as specified by the combination of encryption algorithm and encryption key length. The encryption token is generated from the server connection key as specified by the combination of encryption algorithm and encryption key length.

The application requester encrypts the user ID and the password using the encryption seed and the encryption token. The encrypted user ID and password are sent in two SECTKN instance variables on the SECCHK command, user ID SECTKN first, password SECTKN second. The application server decrypts the user ID and password using the encryption seed and the encryption token. The plain text user ID and password are then passed to the local security server to be authenticated.

On successful connection, if the request data stream contains security-sensitive objects, then the application requester encrypts the DSS carrier for the security-sensitive objects using the encryption token and the encryption seed generated during connect processing. The application requester sets the *dsstype*, Encrypted OBJDSS to indicate that the object is encrypted. The application server decrypts the encrypted objects using the encryption token and the encryption seed generated during connect processing.

If the reply data stream contains security-sensitive objects, then the application server encrypts the DSS carrier for the security-sensitive objects using the encryption token and the encryption seed. The application server sets the *dsstype*, Encrypted OBJDSS to indicate that the object is encrypted. The application requester decrypts the encrypted objects using the encryption token and the encryption seed.

See the following terms for the specific meaning and function of the security mechanism:

DHENC	Diffie-Hellman Encryption Security Mechanism (see <i>DHENC</i> (on page 280))
EDTASECOVR	Encrypted User ID Password and Data Security Overview (see <i>EDTASECOVR</i> (on page 323))
EUSRIDDTA	Encrypted User ID and Encrypted Security-sensitive Data (see <i>EUSRIDDTA</i> (on page 355))
EUSRNPWDDTA	Encrypted User ID, Encrypted Password, Encrypted New Password, and Encrypted Security-sensitive Data (see <i>EUSRNPWDDTA</i> (on page 359))
SECTKNOVR	Security Token Override (see <i>SECTKNOVR</i> (on page 822))

---

value	12
-------	----

**SEE ALSO**

<b>insvar</b>	<i>ACCSEC</i> (on page 52) <i>ACCSECRD</i> (on page 58) <i>SECCHK</i> (on page 800) <i>SECMEC</i> (on page 811) <i>SECTKNOVR</i> (on page 822)
<b>Semantic</b>	<i>DHENC</i> (on page 280) <i>EDTASECOVR</i> (on page 323) <i>EUSRIDDTA</i> (on page 355) <i>EUSRNPWDDTA</i> (on page 359) <i>SECCHK</i> (on page 800) <i>SECMEC</i> (on page 811)



**NAME**

EXCSAT — Exchange Server Attributes

**DESCRIPTION (Semantic)****Dictionary** QDDBASD**Codepoint** X'1041'**Length** \***Class** CLASS**Sprcls** COMMAND - Command

The Exchange Server Attribute (EXCSAT) command exchanges the following information between servers:

- The server's class name
- The architectural level of each class of managers it supports
- The server's product release level
- The server's external name
- The server's name

A server is an instance of DDM architecture implementation plus any product-specific extensions to the architecture. The AS/400 file server is an example of a server class. Managers are the objects of a server supporting and managing various classes of data storage, organization, and access. Files, directories, and dictionaries are examples of managers (see *SERVER* (on page 824) and *MANAGER* (on page 491)).

**Source System Processing**

The source server sends EXCSAT to the supervisor of the remote server to identify itself and its capabilities.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**DSS Carrier: RQSDSS****Target System Processing**

The target server responds by returning an EXCSATRD to identify itself and its capabilities to the source server. If the source system does not send any of the optional parameters of EXCSAT, then the target system must not return the optional parameters in EXCSATRD. Except for the *mgrlvlls* parameter, the target system is not obliged to return any of the optional parameters of the EXCSATRD.

### Exchanging Information

The following sections describe the reasons and methodology for exchanging server information between the source and target servers.

### Exchanging Server Class Names

Exchanging server class names between source and target servers allows each server to determine which codepoint dictionaries the other server has available. Each codepoint dictionary is assigned a fixed index value in the list of dictionaries known to all servers, but not all dictionaries are supported by all classes of servers. For example, AS/400 supports the AS/400 extension dictionary but may not support another system's extension dictionary.

Each server must examine the server class name of its communications partner and determine if there is a predefined agreement on the dictionaries they both support. If there is not, only codepoints of classes the DDM architecture dictionaries (QDDPRMD, QDDBASD, and QDDRDBD) define can be used. Because cross-product connectivity is the goal of DDM, all products must be able to support connections based only on codepoints from the DDM architecture dictionaries. There are no other architectural restraints on the development of homogeneous product or product-pair agreements.

The target server selects either the predetermined dictionary list or the list of only DDM architecture dictionaries. The source server selects either the predetermined dictionary list, or the list of only DDM architecture dictionaries, or terminates communications with the target server.

### Exchanging Manager-Level Lists

Each class of managers (those classes with a superclass of MANAGER) DDM defines has a *mgrlvln* parameter defined in its class description. As the architected definition of a manager class evolves over time, higher-level numbers are sequentially assigned to its *mgrlvln* parameter. DDM requires higher levels of DDM to support all functions and capabilities of lower levels, but the inverse might not be true (this also can be false for product extensions). For example, a Level 3 direct file class must support all functions and capabilities of level 2 and Level 1 direct files, but Level 1 direct files are not required to support all functions and capabilities of Level 2 or Level 3 direct files.

When a source server sends an EXCSAT command, it can specify the *mgrlvlls* parameter. Each entry consists of a manager class code point and the level of support the source requests. When a target server receives an EXCSAT command with a *mgrlvlls* parameter, it must return an EXCSATRD with a *mgrlvlls* parameter. For each manager the source (including duplicates) specifies, the target must return the level of its support for that manager. The target server must not provide this information for any target managers unless the source explicitly requests.

For each manager class, if the target server's support level is greater than or equal to the source server's level, then the source server's level is returned for that class if the target server can operate at that source's level; otherwise, a level of 0 is returned. If the target server's support level is less than the source server's level, the target server's level is returned for that class. If the target server does not recognize the codepoint of a manager class or does not support that class, it returns a level of 0. The target server then waits for the next command or for the source server to terminate communications.

When the source server receives EXCSATRD, it must compare each of the entries in the *mgrlvlls* parameter it received to the corresponding entries in the *mgrlvlls* parameter it sent. If any level mismatches, the source server must decide whether it can use or adjust to the lower level of target support for that manager class. There are no architectural criteria for making this

decision. The source server can terminate communications or continue at the target server's level of support. It can also attempt to use whatever commands its user requests while receiving error reply messages for real functional mismatches.

The manager levels the source server specifies or the target server returns must be compatible with the manager-level dependencies of the specified managers. In other words, incompatible manager levels cannot be specified. For example, specifying the STRAM manager at Level 2 and the LCKMGR at Level 1 introduces an incompatibility because the Level 2 STRAM manager depends on the Level 2 LCKMGR. The dependencies that managers have on each other are specified by the **mgrdepls** parameter of each class of manager.

### Using the CCSIDMGR

When the CCSIDMGR is specified in the *mgrlvlls* parameter, the manager-level number is redefined. Instead of a level number, the value of the sender's CCSID is used. If the source server does not specify the CCSIDMGR in the *mgrlvlls* parameter, all character data in the parameters of the DDM commands and reply messages are in EBCDIC CCSID 500. If the target server does not support the CCSIDMGR, the manager level number value returned in the EXCSATRD is zero. This is consistent with the normal processing of the *mgrlvlls* parameter. If the target server supports the CCSIDMGR but not the CCSID specified in the manager-level number value from the source, the target must return a value of minus one (FFFF) in the manager-level number.

### Exchanging Server Release Levels

The source server can send the *svrslv* parameter on the EXCSAT command to provide the target server with information about the source's product release level. The target server can return the *svrslv* parameter on the EXCSATRD to provide the source server with information about the target's product release level. Because the server release levels are defined to be unarchitected strings, this information is likely to be useful only when the source and target server are of the same server class. There is no architected use for this information, and either the source or the target server can ignore it.

### Exchanging Server External Names

The source server can send the *extnam* parameter on the EXCSAT command to provide the target server with the name of the source system job (or task) it is servicing. The target server can return the *extnam* parameter on the EXCSATRD to provide the source server with the name of the target system job (or task) that will provide DDM services. External names are unarchitected character strings used when logging messages regarding communications failures or DDM processing failures. There is no architected use for this information, and either the source or the target server can ignore it.

### Exchanging Server Names

The source server optionally can send the *svnam* parameter on the EXCSAT command to provide the target server with the name of the source system server, itself. The target server optionally can return the *svnam* parameter on the EXCSATRD to provide the source server with the name of the target system server that provides DDM services. Server names are servers that will provide DDM services. Server names are unarchitected character strings used when logging messages regarding communications failures or DDM processing failures. There is no architected use for this information, and either the source or the target can ignore it.

### Handling Subsequent EXCSAT Commands

The first command the source sends to the target must be the EXCSAT command. It can also be sent additional times, but the following considerations apply to subsequent sendings:

- The target must ignore the values the source specifies for the *extnam*, *svclsnm*, *srvnam*, and *svrslsv* parameter. If any of these parameters are specified, the target must return in EXCSATRD the same values that were returned (or would have been returned) by the first EXCSAT command.
- The manager levels the source specifies in the *mgrlvlls* parameter can only add new managers to those specified in the first EXCSAT command. The source cannot respecify manager levels. Further, the dependencies of the new managers must be compatible with those of the previously specified managers.

If any of these rules are violated, the MGRLVLRM is returned.

### Data Conversion

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

### Effect on Cursor Position

The cursor position of any file that is open when the EXCSAT command is executed does not change.

### Exceptions

The EXCSAT command must be the first command the source sends to the target, or the PRCCNVRM is returned. It can also be sent additional times.

If incompatible manager levels are specified in the *mgrlvlls* parameter, or if an attempt to respecify a manager level is made, the command is rejected with the MGRLVLRM.

### Source System Reply Processing

Return any reply messages and data objects to the requester.

### Data Conversion

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'1041'	
<b>extnam</b>	INSTANCE_OF	EXTNAM - External Name
	OPTIONAL	
	IGNORABLE	

mgrlvlsls	INSTANCE_OF OPTIONAL	MGRVLVLS - Manager-level List
spvnam	INSTANCE_OF OPTIONAL NOTE  CMDTRG	SPVNAM - Supervisor Name  The value of this parameter does not have to be validated.
srvclsnm	INSTANCE_OF OPTIONAL NOTE	SRVCLSNM - Server Class Name  The value of this parameter does not have to be validated.
srvnam	INSTANCE_OF OPTIONAL IGNORABLE	SRVNAM - Server Name
srvrslsv	INSTANCE_OF OPTIONAL IGNORABLE	SRVRLSLV - Server Product Release Level
clscmd	NIL	
inscmd	NIL	
cmddta	NIL	
<b>rpydta</b>		<b>REPLY OBJECTS</b>
X'1443'	INSTANCE_OF REQUIRED	EXCSATRD - Server Attributes Reply Data
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported
X'1218'	INSTANCE_OF	MGRDEPRM - Manager Dependency Error
X'1210'	INSTANCE_OF MINLVL	MGRVLVRM - Manager-level Conflict 2
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limits Reached
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'125F'	INSTANCE_OF	TRGNSPRM - Target Not Supported
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

## SEE ALSO

<b>inscmd</b>	<i>SUPERVISOR</i> (on page 908)
<b>insvar</b>	<i>DEPERRCD</i> (on page 270) <i>MGRLVL</i> (on page 506) <i>MGRLVLRM</i> (on page 512) <i>PRCCNVCD</i> (on page 621)
<b>Semantic</b>	<i>ACCRDB</i> (on page 42) <i>ACCSEC</i> (on page 52) <i>BNDOPTNM</i> (on page 134) <i>BNDOPTVL</i> (on page 135) <i>CCSIDMGR</i> (on page 150) <i>CMNAPPC</i> (on page 184) <i>CMNSYNCPT</i> (on page 202) <i>CMNTCPIP</i> (on page 214) <i>DCESECOVR</i> (on page 253) <i>EDTASECOVR</i> (on page 323) <i>EXCSATRD</i> (on page 369) <i>EXTENSIONS</i> (on page 400) <i>KERSECPPL</i> (on page 470) <i>LVLCMP</i> (on page 486) <i>MGRLVLN</i> (on page 509) <i>MGRLVLOVR</i> (on page 511) <i>MGRLVLRM</i> (on page 512) <i>PLGINNM</i> (on page 614) <i>PLGINOVR</i> (on page 615) <i>PLGINPPL</i> (on page 619) <i>SECMGR</i> (on page 814) <i>SNDPKT</i> (on page 829) <i>SUPERVISOR</i> (on page 908) <i>SYNCMNI</i> (on page 931) <i>SYNCPTOV</i> (on page 944) <i>TCPCMNI</i> (on page 994) <i>USRSECOVR</i> (on page 1050)

**NAME**

EXCSATRD — Server Attributes Reply Data

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD  
**Codepoint** X'1443'  
**Length** \*  
**Class** CLASS  
**Sprcls** COLLECTION - Collection Object

The Server Attributes Reply Data (EXCSATRD) returns the following information in response to an EXCSAT command:

- The target server's class name
- The target server's support level for each class of manager the source requests (see *EXCSAT* (on page 363), *SERVER* (on page 824), and *MANAGER* (on page 491))
- The target server's product release level
- The target server's external name
- The target server's name

**DSS Carrier: OBJDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'1443'	
<b>extnam</b>	INSTANCE_OF OPTIONAL	EXTNAM - External Name
<b>mgrlvlsl</b>	INSTANCE_OF OPTIONAL	MGRLVLLS - Manager-level List
<b>srvclsnm</b>	INSTANCE_OF OPTIONAL	SRVCLSNM - Server Class Name
<b>srvnam</b>	INSTANCE_OF OPTIONAL	SRVNAM - Server Name
<b>srvrlslv</b>	INSTANCE_OF OPTIONAL	SRVRLSLV - Server Product Release Level
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *PRCCNVCD* (on page 621)  
**rpydta** *EXCSAT* (on page 363)  
**Semantic** *CCSIDMGR* (on page 150)  
*DCESECOVR* (on page 253)  
*EDTASECOVR* (on page 323)  
*EXCSAT* (on page 363)

*MGRLVLOVR* (on page 511)  
*PLGINOVR* (on page 615)  
*SYNCPTOV* (on page 944)  
*TCPCMNI* (on page 994)  
*USRSECOVR* (on page 1050)



**NAME**

EXCSQLIMM — Execute Immediate SQL Statement

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'200A'**Length** \***Class** CLASS**Sprcls** COMMAND - Command

The Execute Immediate SQL Statement (EXCSQLIMM) Command executes the non-cursor Structured Query Language (SQL) statement sent as command data.

The SQL statement (SQLSTT) sent as command data cannot contain references to either input variables or output variables. The relational database (RDB) can limit the list of SQLSTTs that can be executed by using the EXCSQLIMM command.

**Source System Processing**

The source system determines the location of the RDB:

Local            Call the local RDB server.

Remote          Send the EXCSQLIMM command to the remote RDB server.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**DSS Carrier: RQSDSS****Target System Processing**

The optional *cmdsrid* parameter uniquely identifies the source of the command.

The *monitor* parameter allows the requester to request a set of events or activities to be monitored on the target server. Refer to MONITOR for the list of supported events.

The *pkgnamcsn* parameter specifies the fully qualified package name, the package's consistency token, and a section number within the package being used to execute the SQL statement.

The *qryinsid* parameter may also be required if the SQL statement being executed is a positioned delete/update. In this case, the *cmdsrid* and the cursor name as specified on the SQL statement text are used to uniquely identify a query. The *qryinsid*, which is mandatory if more than a single query instance exists for the query, is then used to uniquely identify an instance of this query.

The *rdcbmtok* parameter specifies whether the RDB should allow the processing of commit or rollback operations.

The *rdbnam* parameter specifies the name of the RDB to which the ACCRDB command gained access. If the *rdbnam* parameter is specified, its value must be the same as the *rdbnam* parameter specified on the ACCRDB command.

The optional *rtsetsstt* parameter specifies whether upon successful processing of the command the target server must return one or more SQLSTT reply data objects, each containing an SQL SET statement for a special register whose settings has been modified on the current connection, if any special register has had its setting modified during execution of the command.

The TYPDEFNAM command data object specifies the name of the data type to data representation mapping definitions that the source SQLAM uses when sending command data objects for this command.

The TYPDEFOVR command data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type to data representation mapping definitions for the command data objects for this command.

The EXCSQLIMM command is followed by an SQLSTT command data object which is the SQL statement being executed.

The SECTKNOVR command data object is sent by the intermediate server, if the SQLSTT command data object is encrypted. The SECTKNOVR contains the encryption seed and the encryption token used to encrypt the SQLSTT object. The SECTKNOVR DSS is encrypted and must be sent prior to the encrypted SQLSTT command data object.

Normal completion of the EXCSQLIMM command returns an SQLCARD object. An RDBUPDRM must be returned if the command makes recoverable updates in the RDB, and the server has not returned a prior RDBUPDRM in the current unit of work. A recoverable update is an RDB update that writes information to the RDB's recovery log.

The TYPDEFNAM reply data object specifies the name of the data type to data representation mapping definitions that the target SQLAM uses when sending the SQLCARD object for this command.

The TYPDEFOVR reply data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type to data representation mapping definitions for the SQLCARD reply data object for this command.

One or reply data objects may also be returned following the SQLCARD reply data object as per the setting of the *rtsetsstt* parameter.

### **Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

### **Exceptions**

Exception conditions detected by the RDB are reported in an SQLCARD object.

If the target SQLAM detects that the value of the SQLSTT command data object does not have the characteristics that the FD:OCA descriptor claims, then the command is rejected with the DTAMCHRM.

If the bind process is active, then the command is rejected with the PKGBPARM.

If the executed SQL statement causes the unit of work to be committed or rolled back, then the ENDUOWRM is sent prior to any reply data objects.

If access to the specified RDB is not in effect, then the command is rejected with the RDBNACRM.

If a commit or rollback operation is attempted at the RDB, then the command is rejected with the CMMRQSRM<sup>8</sup> unless *rdcbmtok* is set to TRUE.

### Source System Reply Processing

Return any reply messages and data objects to the requester.

### Data Conversion

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

### Package Name and Consistency Token

With SQLAM level 7, the PKGNAMCSN is changed to an optional instance variable. If specified, the values are used as the default package name and consistency token for the connection. If not specified on the command, the PKGSN is required to identify the section number. In this case, the default package name, consistency token, and specified section number are used for the request. If PKGNAMCSN was never specified on the connection, a PRCCNVRM is returned.

clsvar	NIL	
insvar	CLASS INSTANCE VARIABLES	
length	*	
class	X'2005'	
cmdsrcid	INSTANCE_OF OPTIONAL MINLVL	CMDSRCID - Command Source Identifier 7
monitor	INSTANCE_OF OPTIONAL MINLVL	MONITOR - Monitor Events 7
pkgnamcsn	INSTANCE_OF  MINLVL OPTIONAL MTLEXC	PKGNAMCSN - RDB Package Name, Consistency Token, and Section Number 7 Required if PKGSN not specified. PKGSN
pkgsn	INSTANCE_OF MINLVL OPTIONAL MTLEXC	PKGSN - RDB Package Section Number 7 Required if PKGNAMCSN not specified. PKGNAMCSN
qryinsid	INSTANCE_OF OPTIONAL REQUIRED	QRYINSID - Query Instance Identifier  If the EXCSQLIMM command is operating on a positioned delete/update SQL statement, and more than a single query instance exists for the section

8. This reply message was added in DDM Level 4.

		associated with the query.
	MINLVL	7
rdbcmtok	INSTANCE_OF OPTIONAL	RDBCMTOK - RDB Commit Allowed
	MINLVL	5
	DFTVAL	X'F0' - FALSE - False State
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
rtnsetstt	INSTANCE_OF OPTIONAL	RTNSETSTT - Return SET Statement
	MINLVL	7
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>cmddta</b>		<b>COMMAND OBJECTS</b>
X'002F'	INSTANCE_OF	TYPDEFNAM - Data Type Definition Name
	ENUVAL	'QTDSQL370'
	ENUVAL	'QTDSQL400'
	ENUVAL	'QTDSQLX86'
	ENUVAL	'QTDSQLASC'
	MINLVL	4
	ENUVAL	'QTDSQLVAX'
	MINLVL	4
	OPTIONAL	
	DFTVAL	''
	NOTE	The default means the value received on ACCRDB command is used.
X'0035'	INSTANCE_OF OPTIONAL	TYPDEFOVR - TYPDEF Overrides
	DFTVAL	''
	NOTE	The default means the value received on ACCRDB command is used.
X'190B'	INSTANCE_OF OPTIONAL	SECTKNOVR - SECTKN Override
	NOTE	This is sent by the intermediate server, if the SQLSTT command data object is encrypted. This must precede the SQLSTT command data object.
X'2414'	INSTANCE_OF REQUIRED	SQLSTT - SQL Statement
<b>rpydta</b>		<b>REPLY OBJECTS</b>
X'002F'	INSTANCE_OF	TYPDEFNAM - Data Type Definition Name
	ENUVAL	'QTDSQL370'
	ENUVAL	'QTDSQL400'
	ENUVAL	'QTDSQLX86'
	ENUVAL	'QTDSQLASC'
	MINLVL	4

	ENUVAL	'QTDSQLVAX'
	MINLVL	4
	OPTIONAL	
	DFTVAL	''
	NOTE	The default means the value received on ACCRDBRM is used.
X'0035'	INSTANCE_OF OPTIONAL DFTVAL NOTE	TYPDEFOVR - TYPDEF Overrides  '' The default means the value received on ACCRDBRM is used.
X'1C00'	INSTANCE_OF OPTIONAL MINLVL	MONITORRD - Monitor Reply Data  7
X'1C03'	INSTANCE_OF OPTIONAL MINLVL	MGRLVLOVR - Manager Level Overrides  7
X'2408'	INSTANCE_OF REQUIRED	SQLCARD - SQL Communications Area Reply Data
X'2414'	INSTANCE_OF OPTIONAL REPEATABLE MINLVL	SQLSTT - SQL Statement   7
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'220D'	INSTANCE_OF	ABNUOWRM - Abnormal End Unit of Work Condition
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported
X'2225'	INSTANCE_OF MINLVL	CMMRQSRM - Commitment Request  4
X'220E'	INSTANCE_OF	DTAMCHRM - Data Descriptor Mismatch
X'220C'	INSTANCE_OF	ENDUOWRM - End Unit of Work Condition
X'1253'	INSTANCE_OF	OBJNSPRM - Object Not Supported
X'2209'	INSTANCE_OF	PKGBPARM - RDB Package Binding Process Active
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'2204'	INSTANCE_OF	RDBNACRM - RDB Not Accessed
X'2218'	INSTANCE_OF MINLVL	RDBUPDRM - RDB Update Reply Message  4
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limits Reached
X'2213'	INSTANCE_OF	SQLERRRM - SQL Error Condition
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'125F'	INSTANCE_OF	TRGNSPRM - Target Not Supported

---

X'1252'      INSTANCE\_OF      VALNSPRM - Parameter Value Not Supported

**SEE ALSO**

**inscmd**      *SQLAM* (on page 847)

**Semantic**      *APPSRCCD* (on page 79)  
*BGNBND* (on page 110)  
*CMDSRCID* (on page 178)  
*OPNQRV* (on page 555)  
*RDBALWUPD* (on page 726)  
*SECTKNOVR* (on page 822)  
*SQL* (on page 835)  
*SQLAM* (on page 847)  
*SYNCPTOV* (on page 944)  
*TCPSRCCD* (on page 1007)

**NAME**

EXCSQLSET — Set SQL Environment

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2014'  
**Length** \*  
**Class** CLASS  
**Sprcls** COMMAND - Command

The Execute SQL Set command executes a list of SQL SET statements at the RDB previously set by the application. SQL SET statements allow the application to influence how SQL statements are executed by the RDB. This command allows the application when connected to multiple RDBs to maintain a consistent execution environment across all connections without having to send individual SQL SET statements to each RDB. The command allows the source system to manage SET information established by the application and propagate the changes to the target server prior to the execution of any SQL statements at the target server.

**Source System Processing**

SET statements are sent to the target server with this command as data objects. The source system maintains a list of non-local SQL SET statements processed by the application. Prior to processing any SQL statements at a target server, the source system propagates these SET statements to the target server by sending this command.

SQL SET statements are sent as a list of SQLSTT data objects. They cannot contain any references to either input or output variables.

The SECTKNOVR command data object is sent by the intermediate server, if the SQLSTT command data object is encrypted. The SECTKNOVR contains the encryption seed and the encryption token used to encrypt the SQLSTT object. The SECTKNOVR DSS is encrypted and must be sent prior to the encrypted SQLSTT command data object.

The *monitor* parameter allows the requester to request a set of events or activities to be monitored on the target server. Refer to MONITOR for the list of supported events.

The *pkgnamct* parameter specifies the fully qualified name and the consistency token of the package being executed.

The *rdnam* parameter specifies the name of the RDB to which the ACCRDB command gained access.

The optional *rtsetstt* parameter specifies whether upon successful processing of the command the target server must return one or more SQLSTT reply data objects, each containing an SQL SET statement for a special register whose settings has been modified on the current connection, if any special register has had its setting modified during execution of the command.

The TYPDEFNAM command data object specifies the name of the data type-to-data representation mapping definitions that the source SQLAM uses when sending command data objects for this command.

The TYPDEFOVR command data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type-to-data representation mapping definitions for the data objects for this command.

Normal completion of this command returns an SQLCARD object. If the SQLCARD describes a successful completion of the command or a warning occurred during the executing of the command, the environment used to process subsequent SQL statements is established. Warnings should not impact the execution of subsequent SQL statements. If the SQLCARD describes an error, the processing of subsequent SQL statements by the target server is prevented until the execution of this command is successful.

One or more SQLSTT reply data objects may also be returned following the SQLCARD reply data object as per the setting of the *rtsetsstt* parameter.

### **Target System Processing**

The RDB is invoked to execute each SET statement in the order that the SQLSTT data objects are received. If the RDB does not recognize the SET statement, a warning SQLCARD data object is returned. If the RDB detects an error condition that should prevent the execution of subsequent SQL statements, the SQLERRRM reply message is returned with an error SQLCARD data object.

### **DSS Carrier: RQSDSS**

### **Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

### **Exceptions**

Unrecognized SET statements the RDB detects are returned as warning conditions in an SQLCARD reply data object.

Exception conditions the RDB detects are reported in an SQLERRRM reply message with an SQLCARD reply data object.

If the target SQLAM detects the value of the SQLSTT command data object and it does not have the characteristics that the FD:OCA descriptor claims, the command is rejected with the DTAMCHRM reply message.

If the bind process is active, the command is rejected with the PKGBPARM reply message.

If access to the specified RDB is not in effect, the command is rejected with the RDBNACRM reply message.

### **Package Name and Consistency Token**

With SQLAM level 7, the PKGNAMCT is changed to an optional instance variable. If specified, the values are used as the default package name and consistency token for the connection. If not specified on the command, the default package name and consistency token are used for the request. If PKGNAMCSN was never specified on the connection, a PRCCNVRM is returned.



## Summary Decision Table

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'2005'	
monitor	INSTANCE_OF OPTIONAL MINLVL	MONITOR - Monitor Events 7
pkgnamct	INSTANCE_OF MINLVL OPTIONAL	PKGNAMECT - RDB Package Name and Consistency token 7
pkgsn	INSTANCE_OF MINLVL OPTIONAL MTLEXC	PKGSN - RDB Package Section Number 7 Required if PKGNAMECT not specified. PKGSN
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
rtnsetstt	INSTANCE_OF OPTIONAL MINLVL	RTNSETSTT - Return SQL Statement 7
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>cmddta</b>	<b>COMMAND OBJECTS</b>	
X'190B'	INSTANCE_OF OPTIONAL NOTE	SECTKNOVR - SECTKN Override  This is sent by the intermediate server, if the SQLSTT command data object is encrypted. This must precede the SQLSTT command data object.
X'002F'	INSTANCE_OF NOTE	TYPDEFNAM - Data Type Definition Name Unchanged, except may not be sent for EXTDTA objects.
X'0035'	INSTANCE_OF	TYPDEFOVR - TYPDEF Overrides Unchanged.
<b>rpydta</b>	<b>REPLY OBJECTS</b>	
X'002F'	INSTANCE_OF NOTE	TYPDEFNAM - Data Type Definition Name Unchanged, except ignored if sent for EXTDTA objects.
X'0035'	INSTANCE_OF NOTE	TYPDEFOVR - TYPDEF Overrides Unchanged, except ignored if sent for EXTDTA objects.
X'1C00'	INSTANCE_OF OPTIONAL MINLVL	MONITORRD - Monitor Reply Data 7

X'1C03'	INSTANCE_OF OPTIONAL MINLVL	MGRLVLOVR - Manager Level Overrides 7
X'2408'	INSTANCE_OF REQUIRED	SQLCARD - SQL Communications Area Reply Data
T}		
X'2414'	INSTANCE_OF OPTIONAL REPEATABLE MINLVL	SQLSTT - SQL Statement 7
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'220D'	INSTANCE_OF	ABNUOWRM - Abnormal End Unit of Work Condition
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported
X'220A'	INSTANCE_OF	DSCINVRM - Invalid Descriptor
X'220E'	INSTANCE_OF	DTAMCHRM - Data Descriptor Mismatch
X'220C'	INSTANCE_OF	ENDUOWRM - End Unit of Work Condition
X'1253'	INSTANCE_OF	OBJNSPRM - Object Not Supported
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'2204'	INSTANCE_OF	RDBNACRM - RDB Not Accessed
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limit Reached
X'2213'	INSTANCE_OF	SQLERRRM - SQL Error Condition
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'125F'	INSTANCE_OF	TRBNSPRM - Target Not Supported
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

**SEE ALSO**

**Semantic**      *RTNSETSTT* (on page 794)  
*SECTKNOVR* (on page 822)

**NAME**

EXCSQLSTT — Execute SQL Statement

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'200B'**Length** \***Class** CLASS**Sprcls** COMMAND - Command

The Execute SQL Statement (EXCSQLSTT) command executes a non-cursor Structured Query Language (SQL) statement that was previously bound into a named package of a relational database (RDB). The SQL statement can optionally include references to input variables, output variables, or both.

An EXCSQLSTT command may or may not be an atomic operation when the operation is multi-row input. It can also be part of an atomic chain of EXCSQLSTT commands. The atomicity of an EXCSQLSTT command for a multi-row input operation is specified by the optional Atomicity Indicator (ATMIND) instance variable.

**Source System Processing**

The source system determines the location of the RDB:

**Local** Call the local RDB server.

**Remote** Send the EXCSQLSTT command to the remote RDB server.

If there are LOBs in the output, the following applies.

An OUTOVR object may be sent as command data for an EXCSQLSTT if the statement returns output and is not a stored procedure call.

If the EXCSQLSTT command is for a stored procedure call or if the command does not return output data, then the OUTOVR object is rejected by the application server with an OBJNSPRM.

If the OUTOVR object is valid for the command, then the descriptor specifies the format in which the output is to be returned. It is required if the output includes LOB data columns that are to be returned in a format other than LOB data.

For a stored procedure call with output parameters, the SQLDTA object contains the output parameter description and use of OUTOVR object for this purpose is incorrect.

If the stored procedure call can return query result sets, then an OUTOVR object may be sent with CNTQRY commands to retrieve data in a result set. The source system specifies when an OUTOVR object may be sent with the CNTQRY command by means of the *outovropt* parameter of the EXCSQLSTT.

**DSS Carrier: RQSDSS****Target System Processing**

The *atmind* parameter specifies the atomicity of the EXCSQLSTT command.

The optional *cmdsrcid* parameter uniquely identifies the source of the command.

The *monitor* parameter allows the requester to request a set of events or activities to be monitored on the target server. Refer to *MONITOR* (on page 521) for the list of supported events.

The *nbrrow* parameter specifies the number of rows to insert if the EXCSQLSTT's target operation is an insert or a multi-row input. If the value of this parameter does not match the number of input data rows, except when the last row for the operation has a null indicator value of -3, as detailed in Section 5.5.3.1 of the DRDA Reference, the application server responds with PRCCNVRM with a *prccnvcd* value of X'1E'.

The *outexp* parameter indicates whether the requester expects the target SQLAM to return output within an SQLDTARD reply data object as a result of the execution of the referenced SQL statement. See *OUTEXP* (on page 571) for more explanation.

The *pkgnamcsn* parameter specifies the fully qualified package name, the package's consistency token, and a section number within the package containing the SQL statement being executed.

The *prcnam* parameter specifies the stored procedure name. This name must follow the target system's semantic and syntactic rules for procedure names. See *PRCNAM* (on page 627) for usage rules.

The *qryinsid* parameter may also be required if the SQL statement being executed is a positioned delete/update. In this case, the *cmdsrcid* and the cursor name as specified on the SQL statement text are used to uniquely identify a query. The *qryinsid*, which is mandatory if more than a single query instance exists for the query, is then used to uniquely identify an instance of this query.

The *qyrowset* parameter specifies a DRDA rowset size for any non-scrollable, non-rowset result sets conforming to the limited block query protocol and for any non-dynamic scrollable, non-rowset result sets that may be returned by the stored procedure. If this is not a stored procedure call or if the stored procedure call returns no such result sets, then this parameter is ignored. For more information on scrollable cursors, see the DRDA Reference, Appendix B.

The *rdbcmtok* parameter specifies whether the RDB should allow the processing of commit or rollback operations.

The *rdbnam* parameter specifies the name of the RDB to which the ACCRDB command gained access. If the *rdbnam* parameter is specified, its value must be the same as the *rdbnam* parameter specified on the ACCRDB command.

The optional *rtissett* parameter specifies whether upon successful processing of the command the target server must return one or more SQLSTT reply data objects, each containing an SQL SET statement for a special register whose settings has been modified on the current connection, if any special register has had its setting modified during execution of the command. If the EXCSQLSTT command is part of an atomic compound chain, the *rtissett* setting as specified on the BGNATMCHN command takes precedence.

If the source SQLAM does not support scrollable cursor capability, but the target SQLAM does, then the following applies:

- All stored procedure result sets that are scrollable revert to non-scrollable result sets. The target SQLAM ensures the cursor is terminated when the end of the query is reached

(SQLSTATE of 02000).

- At intermediate sites, the intermediate site fails the EXCSQLSTT with SQLSTATE 560B3 if it cannot return scrolling parameters received on the OPNQRYRM because the upstream site is downlevel. Thus, only the target server can revert a scrollable cursor to non-scrollable, and each intermediate site must accurately reflect the server's decision if it can.

The TYPDEFNAM command data object specifies the name of the data type to data representation mapping definitions that the source SQLAM uses when sending command data objects for this command.

The TYPDEFOVR command data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type to data representation mapping definitions for the command data objects for this command.

If the SQL statement being executed is not a CALL statement or other statement that invokes a stored procedure and the SQL statement has input host variables, an SQLDTA command data object is sent following the command to describe and carry the input host variable data values.

If the SQL statement being executed is a CALL statement or other statement that invokes a stored procedure and the SQL statement has host variables in the parameter list, an SQLDTA command data object is sent following the command to describe and carry the host variable data values. If the CALL or other statement that invokes the stored procedure specifies the procedure name using a host variable, then the PRCNAM parameter specifies the procedure name value.

The SECTKNOVR command data object is sent by the intermediate server, if the SQLDTA command data object is encrypted. The SECTKNOVR contains the encryption seed and the encryption token used to encrypt the SQLDTA object. The SECTKNOVR DSS is encrypted and must be sent prior to the encrypted SQLDTA command data object.

Normal completion of this command returns an SQLDTARD or SQLCARD object. An RDBUPDRM must be returned if the command makes recoverable updates in the RDB, and the server did not return a prior RDB in the current unit of work. A recoverable update is an RDB update that writes information to the RDB's recovery log.

One or more SQLSTT reply data objects may also be returned following the SQLCARD or SQLDTARD reply data object as per the setting of the *rtissetstt* parameter.

The SECTKNOVR reply data object is sent by the intermediate server, if the SQLDTARD reply data object is encrypted. The SECTKNOVR contains the encryption seed and the encryption token used to encrypt the SQLDTARD object. The SECTKNOVR DSS is encrypted and must be sent prior to the encrypted SQLDTARD reply data object.

The TYPDEFNAM reply data object specifies the name of the data type to data representation mapping definitions that the target SQLAM uses when sending the reply data objects for this command to the source SQLAM.

The TYPDEFOVR reply data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type to data representation mapping definitions for the reply data objects for the EXCSQLSTT command.

Normal completion of this command can return an SQLDARD and/or SQLCINRD reply object. To get an SQLDA descriptor area for the executed SQL statement, the RTNSQLDA, RSLSETFLG, and TYPSQLDA instance variables define if and how the descriptors are returned.

The RTNSQLDA instance variable controls whether to return an SQLDA descriptor area for the output variables for the executed SQL statement. The RSLSETFLG controls what SQLDA descriptor information is returned for each result set returned.

The TYPSQLDA instance variable determines the type of descriptive information to be returned for output variables. Three types of SQLDA descriptor area can be requested: a light, standard, or extended version. Each type provides a different set of descriptive information. The standard SQLDA provides the same descriptive information as was defined for SQLAM level 6 and earlier. The extended SQLDA provides additional descriptive information required by certain types of API such as JDBC.

The SQLDA types are defined by the SQLDARD and SQLCINRD FD:OCA descriptors. Refer to the DRDA Reference for a description of SQLDARD and SQLCINRD.

If the target server ignores the RTNSQLDA parameter, then the target server returns no SQLDARD object to the requester.

### Statements that are not Stored Procedure Calls

If output is expected and an OUTOVR object is received, then the target returns the outputs in the format specified by the OUTOVR. If output is not expected, then the OUTOVR object is not allowed and the EXCSQLSTT command is rejected with an OBJNSPRM.

### Stored Procedures

When an SQL statement invokes a stored procedure, the EXCSQLSTT command may carry additional parameters and may generate result sets. The parameters are:

MAXBLKEXT	Maximum Number of Extra Blocks (see <i>MAXBLKEXT</i> (on page 494))
MAXRSLCNT	Maximum Result Set Count (see <i>MAXRSLCNT</i> (on page 497))
PRCNAM	Procedure Name (see <i>PRCNAM</i> (on page 627))
QRYBLKSZ	Query Block Size (see <i>QRYBLKSZ</i> (on page 678))
QRYROWSET	Query Rowset Size (see <i>QRYROWSET</i> (on page 697))
RSLSETFLG	Result Set Flags (see <i>RSLSETFLG</i> (on page 783))

When a procedure that generates result sets completes, an optional transaction component is returned, followed by a summary component, and finally followed by a query result set component for each result set. The summary component consists of DSSs which indicate the completion status of the stored procedure (SQLCARD or SQLDTARD with one or more SQLSTTs and possibly EXTDTAs associated with the SQLDTARD) and which give information about the result sets (RSLSETRM and SQLRSLRD). The query result set component for each result set includes the required OPNQRYRM reply message, indicating which query protocol applies to the result set, an optional SQLCARD, the optional SQLCINRD reply data object describing the columns in the result set, the required FD:OCA description of the data in the query (see *QRYDSC* (on page 685)), and optional answer set data (see *QRYDTA* (on page 686)). The order that these DSSs are returned and the manner in which they are returned depend on the SQLAM level. Refer to the DRDA Chaining rules (CH) and the DRDA Query Data Transfer Protocol rules (QP) in the DRDA Reference for further information on these rules.

After the EXCSQLSTT or each subsequent CNTQRY command, each result set is suspended unless some condition terminates the result set (see *CNTQRY* (on page 222)). The CNTQRY command can continue a suspended result set so that the next portion of the answer set data is returned. A result set is terminated anytime the CLSQRY command is received and processed.

The SECTKNOVR reply data object is sent by the intermediate server, if the optional reply data objects SQLDTARD, QRYDTA, SQLRSLRD, SQLCINRD, and EXTDTA are encrypted. The SECTKNOVR contains the encryption seed and the encryption token used to encrypt the security-sensitive reply data object. The SECTKNOVR DSS is encrypted and must be sent prior

to the first encrypted reply data object.

The following applies to LOB-related instance variables, command data objects, and reply data objects that may be specified for a stored procedure.

If an OUTOVR object is received with the EXCSQLSTT command, the target rejects the OUTOVR object. This is because stored procedure output parameters are fully specified by the SQLDTA specified with the EXCSQLSTT and the OUTOVR is not applicable.

The *outvropt* parameter specifies the way in which OUTOVR objects can be flowed with CNTQRY commands for any result sets that may be generated by the stored procedure.

If *outvrfrs*, the requester can send an OUTOVR object with the first CNTQRY for the result set. After the first CNTQRY, the target will return an error if an OUTOVR object is sent. If *outvrany*, the requester can send an OUTOVR object with any CNTQRY for the result set.

If no OUTOVR is sent to the target on a CNTQRY for the result set, the target fetches the data using the description of the data given by the last OUTOVR object sent with a CNTQRY command, or, if none was previously sent, the QRYDSC returned with the OPNQRYRM is used.

Data may be returned at the end of the result set description, depending on the query protocol in effect for the result set (see *FIXROWPRC* (on page 417) and *LMTBLKPRC* (on page 475)).

If any EXTDTAs are returned in the EXCSQLSTT command reply chain, then the EXTDTAs must be the last in the chain. In addition, no extra query blocks may be returned for query result sets in this case.

Optionally, an RDBUPDRM reply message may be returned at the end of the reply chain in accordance with the DRDA Update Control (UP) rules (refer to the DRDA Reference for details).

### Data Conversion

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

### Exceptions

Exception conditions the RDB detects are reported in an SQLDTARD or SQLCARD object.

If the bind process is active, then the command is rejected with the PKGBPARM.

If the executed SQL statement causes the unit of work to be committed or rolled back, then the ENDUOWRM is sent prior to any reply data objects.

If the data contained in the *fdodta* parameter of the SQLDTA command data object is not properly described by the Formatted Data Object Content Architecture (FD:OCA) descriptor contained in the *fdodsc* parameter of the SQLDTA command data object, then the command is rejected with the DTAMCHRM.

If the target SQLAM is unable to generate a valid FD:OCA descriptor for the data contained in the SQLDTA command data object, then the command is rejected with the DSCINVRM.

If the specified RDB is not currently accessed, the command is rejected with the RDBNACRM.

If a commit or rollback operation is attempted at the RDB, then the command is rejected with the CMMRQSRM<sup>9</sup> unless *rdcbmtok* is set to TRUE.

If a *prcnam* is specified and the section referred to in *pkgnamcsn* is not associated with a procedure, a CMDCHKRM should be returned to the source server.

If the value of the *nbrrow* parameter does not match the number of input data rows for a multi-row input operation, except when the last row for the operation has a null indicator value of -3, as detailed in Section 5.5.3.1 of the DRDA Reference, the application server responds with PRCCNVRM with a *prccnvc* value of X'1E'.

### Source System Reply Processing

Return any reply messages and data objects to the requester.

### Package Name and Consistency Token

With SQLAM level 7, the PKGNAMCSN is changed to an optional instance variable. If specified, the values are used as the default package name and consistency token for the connection. If not specified on the command, the PKGSN is required to identify the section number. In this case, the default package name, consistency token, and specified section number are used for the request. If PKGNAMCSN was never specified on the connection, a PRCCNVRM is returned.

clsvar	NIL	
insvar	CLASS INSTANCE VARIABLES	
length	*	
class	X'200B'	
atmind	INSTANCE_OF OPTIONAL NOTE DFTVAL MINLVL	ATMIND - Atomicity Indicator  The <i>atmind</i> is REQUIRED if the EXCSQLSTT command is a non-atomic multi-row input operation. X'00' 7
cmdsrcid	INSTANCE_OF OPTIONAL MINLVL	CMDSRCID - Command Source Identifier  7
maxblkext	INSTANCE_OF OPTIONAL DFTVAL	MAXBLKEXT - Maximum Number of Extra Blocks  0
maxrslcnt	INSTANCE_OF OPTIONAL DFTVAL	MAXRSLCNT - Maximum Result Set Count  0
monitor	INSTANCE_OF OPTIONAL MINLVL	MONITOR - Monitor Events  7
nbrrow	INSTANCE_OF	NBRROW - Number of Input Rows

9. This reply message was added in DDM Level 4.



	OPTIONAL NOTE	The <i>nbrrow</i> is REQUIRED with a minimum value of 1 if the EXCSQLSTT command is a multi-row input operation.
	MTLEEXEC NOTE	X'2111' - OUTEXP - Output Expected The NBRROW instance variable is mutually-exclusive with the OUTEXP value of X'F1'.
	MINLVL	4
outexp	INSTANCE_OF OPTIONAL MTLEEXEC NOTE	OUTEXP - Output Expected  X'213A' - NBRROW - Number of Input Rows The OUTEXP value of X'F1' is mutually-exclusive with any NBRROW value.
outovropt	INSTANCE_OF OPTIONAL	OUTOVROPT - Output Override Option
pkgnamcsn	INSTANCE_OF  MINLVL OPTIONAL MTLEXC	PKGNAMECSN - RDB Package Name, Consistency Token, and Section Number 7 Required if PKGSN not specified. PKGSN
pkgsn	INSTANCE_OF MINLVL OPTIONAL MTLEXC	PKGSN - RDB Package Section Number 7 Required if PKGNAMCSN not specified. PKGNAMECSN
prcnam	INSTANCE_OF OPTIONAL NOTE  MINLVL	PRCNAM - Procedure Name  The prcnam is REQUIRED if the procedure name is specified by a host variable. 4
qryblksz	INSTANCE_OF OPTIONAL NOTE  MINLVL	QRYBLKSZ - Query Block Size  The <i>qryblksz</i> is REQUIRED if the EXCSQLSTT command invokes a stored procedure. !MINLVL!5 _ qryinsid!INSTANCE_OF!QRYINSID - Query Instance Identifier !OPTIONAL !REQUIRED!T{ If the EXCSQLSTT command is operating on a positioned delete/update SQL statement, and more than a single query instance exists for the section associated with the query. 7
qryrowset	INSTANCE_OF OPTIONAL IGNORABLE MINLVL	QRYROWSET - Query Rowset Size  If not a stored procedure call. 7
rdbcmtok	INSTANCE_OF OPTIONAL	RDBCMTOK - RDB Commit Allowed

	MINLVL	5
	DFTVAL	X'F0' - FALSE - False State
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
rslsetflg	INSTANCE_OF OPTIONAL DFTVAL	RSLSETFLG - Result Set Flags  B'00000000'
rtnsetstt	INSTANCE_OF OPTIONAL IGNORABLE	RTNSETSTT - Return SET Statement  If the EXCSQLSTT command is part of an atomic chain in which case the RTNSETSTT setting on the BGNATMCHN command takes precedence.
	MINLVL	7
rtnsqlda	INSTANCE_OF OPTIONAL IGNORABLE	RTNSQLDA - Return SQL Descriptor Area
	MINLVL	7
typsqlda	INSTANCE_OF OPTIONAL IGNORABLE	TYPSQLDA - Type of SQL Descriptor  If RTNSQLDA is ignored, TYPSQLDA is ignored as well. Otherwise, TYPSQLDA may not be ignored.
	MINLVL	7
	NOTE	Supported values:
	ENUVAL	0
	NOTE	Request standard output SQLDA.
	ENUVAL	2
	NOTE	Request light output SQLDA.
	ENUVAL	4
	NOTE	Return extended output SQLDA.
	DFTVAL	0
clscmd	NIL	
inscmd	NIL	
cmddta		<b>COMMAND OBJECTS</b>
X'002F'	INSTANCE_OF	TYPDEFNAM - Data Type Definition Name
	ENUVAL	'QTDSQL370'
	ENUVAL	'QTDSQL400'
	ENUVAL	'QTDSQLX86'
	ENUVAL	'QTDSQLASC'
	MINLVL	4
	ENUVAL	'QTDSQLVAX'
	MINLVL	4
	OPTIONAL	
	DFTVAL	''
	NOTE	The default means the value received on the ACCRDB command is used.

	NOTE	May not be sent for EXTDTA objects.
X'0035'	INSTANCE_OF OPTIONAL DFTVAL NOTE	TYPDEFOVR - TYPDEF Overrides " The default means the value received on the ACCRDB command is used.
	NOTE	May not be sent for EXTDTA objects.
X'190B'	INSTANCE_OF OPTIONAL NOTE	SECTKNOVR - SECTKN Override  This is sent by the intermediate server, if the SQLDTA and/or EXTDTA command data objects are encrypted. This must precede the first encrypted command data object.
X'2412'	INSTANCE_OF OPTIONAL NOTE	SQLDTA - SQL Program Variable Data  Specified when the SQL statement has input variables.
X'146C'	INSTANCE_OF	EXTDTA - Externalized FD:OCA Data
X'2147'	INSTANCE_OF NOTE	OUTOVR - Output Override Descriptor Not allowed for SQL statements that are stored procedure calls or if no output is expected.
<b>rpydta</b>		<b>REPLY OBJECTS</b>
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL MINLVL ENUVAL MINLVL OPTIONAL DFTVAL NOTE	TYPDEFNAM - Data Type Definition Name 'QTDSQL370' 'QTDSQL400' 'QTDSQLX86' 'QTDSQLASC' 4 'QTDSQLVAX' 4 " The default means the value received on the ACCRDBRM is used.
	NOTE REPEATABLE	Ignored if sent for EXTDTA objects.
X'0035'	INSTANCE_OF OPTIONAL DFTVAL NOTE	TYPDEFOVR - TYPDEF Overrides " The default means the value received on the ACCRDBRM is used.
	NOTE REPEATABLE	Ignored if sent for EXTDTA objects.
X'190B'	INSTANCE_OF OPTIONAL NOTE	SECTKNOVR - SECTKN Override  This is sent by the intermediate server, if the optional reply data objects SQLDTARD, QRYDTA, SQLRSLRD, SQLCINRD, EXTDTA are encrypted. This must precede the first encrypted reply data object.

X'1C00'	INSTANCE_OF OPTIONAL MINLVL	MONITORRD - Monitor Reply Data 7
X'1C03'	INSTANCE_OF OPTIONAL MINLVL	MGRLVLOVR - Manager Level Overrides 7
X'2408'	INSTANCE_OF OPTIONAL MTLEXC	SQLCARD - SQL Communications Area Reply Data X'2413' - SQLDTARD - SQL Data Reply Data
X'2413'	INSTANCE_OF OPTIONAL MTLEXC  NOTE	SQLDTARD - SQL Data Reply Data  X'2408' - SQLCARD - SQL Communications Area Reply Data The SQLDTARD returns if output results from a non-cursor SQL SELECT statement or if the parameters of an SQL statement that invokes a stored procedure contains host variables.
X'241A'	INSTANCE_OF OPTIONAL REPEATABLE NOTE	QRYDSC - Query Answer Set Description  Returned for each result set generated by the invocation of a stored procedure.
X'241B'	INSTANCE_OF OPTIONAL REPEATABLE NOTE	QRYDTA - Query Answer Set Data  May be returned for each result set generated by the invocation of a stored procedure.
X'240E'	INSTANCE_OF OPTIONAL NOTE	SQLRSLRD - SQL Result Set Reply Data  Returned only if one or more result sets are generated by the invocation of a stored procedure.
X'240B'	INSTANCE_OF OPTIONAL NOTE	SQLCINRD - SQL Result Set Column Information Reply Data May be returned for each result set generated by the invocation of a stored procedure.
X'2414'	INSTANCE_OF OPTIONAL REPEATABLE MINLVL	SQLSTT - SQL Statement 7
X'146C'	INSTANCE_OF	EXTDTA - Externalized FD:OCA Data
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'220D'	INSTANCE_OF	ABNUOWRM - Abnormal End Unit of Work Condition
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check

X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported
X'2225'	INSTANCE_OF MINLVL	CMMRQSRM - Commitment Request 4
X'220A'	INSTANCE_OF	DSCINVRM - Invalid Description
X'220E'	INSTANCE_OF	DTAMCHRM - Data Descriptor Mismatch
X'220B'	INSTANCE_OF MINLVL	ENDQRYRM - End of Query 5
X'220C'	INSTANCE_OF	ENDUOWRM - End Unit of Work Condition
X'1253'	INSTANCE_OF	OBJNSPRM - Object Not Supported
X'2205'	INSTANCE_OF MINLVL NOTE	OPNQRYRM - Open Query Complete 5 Returned only if a stored procedure was called and has completed processing.
X'2209'	INSTANCE_OF	PKGBPARAM - RDB Package Binding Process Active
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'2204'	INSTANCE_OF	RDBNACRM - RDB Not Accessed
X'2218'	INSTANCE_OF MINLVL	RDBUPDRM - RDB Update Reply Message 4
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limits Reached
X'2219'	INSTANCE_OF	RSLSETRM - RDB Result Set Reply Message
X'2213'	INSTANCE_OF	SQLERRRM - SQL Error Condition
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'125F'	INSTANCE_OF	TRGNSPRM - Target Not Supported
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

**SEE ALSO**

<b>insvar</b>	<i>ATMIND</i> (on page 104) <i>ENDATMCHN</i> (on page 333) <i>ENDCHNTYP</i> (on page 341) <i>MAXRSLCNT</i> (on page 497) <i>MONITOR</i> (on page 521) <i>NBRROW</i> (on page 530) <i>PRCCNVCD</i> (on page 621) <i>PRCNAM</i> (on page 627) <i>QRYROWSET</i> (on page 697) <i>RSLSETFLG</i> (on page 783) <i>SYNERRCD</i> (on page 985)
<b>inscmd</b>	<i>SQLAM</i> (on page 847)
<b>Semantic</b>	<i>ATMIND</i> (on page 104) <i>BGNATMCHN</i> (on page 107) <i>BGNBND</i> (on page 110) <i>CMDSRCID</i> (on page 178) <i>CNTQRY</i> (on page 222) <i>EDTASECOVR</i> (on page 323) <i>ENDATMCHN</i> (on page 333) <i>ENDCHNTYP</i> (on page 341)

*FIXROWPRC* (on page 417)  
*LMTBLKPRC* (on page 475)  
*MAXBLKEXT* (on page 494)  
*MAXRSLCNT* (on page 497)  
*MONITOR* (on page 521)  
*OPNQRY* (on page 555)  
*OPNQRYRM* (on page 566)  
*OUTOVR* (on page 572)  
*OUTOVRANY* (on page 574)  
*OUTOVRFRS* (on page 575)  
*PRCCNVRM* (on page 625)  
*PRCNAM* (on page 627)  
*PRPSQLSTT* (on page 636)  
*QRYBLK* (on page 670)  
*QRYBLKFACT* (on page 675)  
*QRYBLKSZ* (on page 678)  
*QRYDTA* (on page 686)  
*QRYROWSET* (on page 697)  
*RDBALWUPD* (on page 726)  
*RDBUPDRM* (on page 751)  
*RSLSETFLG* (on page 783)  
*RSLSETRM* (on page 785)  
*SECTKNOVR* (on page 822)  
*SQL* (on page 835)  
*SQLAM* (on page 847)  
*SQLCINRD* (on page 857)  
*SQLRSLRD* (on page 867)  
*STTASMEUI* (on page 893)

**NAME**

EXPALL — Explain All Explainable SQL Statements

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'243B'**Length** \***Class** codpnt

The Explain All Explainable SQL Statements (EXPALL) command specifies that the target SQLAM causes the target relational database (RDB) manager to explain normally any explainable static SQL statements during the bind or rebind process. Optionally, dynamic SQL statements can be explained at execution time.

**SEE ALSO****insvar** *BNDEXPOPT* (on page 129)**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)

**NAME**

EXPNON — Explain No SQL Statements

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'243A'

**Length** \*

**Class** CODPNT

The Explain No SQL Statements (EXPNON) specifies that the target SQLAM does not cause the target relational database (RDB) to explain any explainable Structured Query Language (SQL) statements during the bind process or at execution time.

**SEE ALSO**

**insvar** *BNDEXPOPT* (on page 129)

**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)



**NAME**

EXPREOPT — Explain All Explainable SQL Statement During Optimization

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2459'**Length** \***Class** CODPNT

Explain All Explainable SQL Statements During Optimization (EXPREOPT) specifies that the target SQLAM causes the target relational database (RDB) manager to explain normally any explainable static and dynamic SQL statements when their access plans are optimized at execution time based on known values of referenced host variables, parameter markers, and/or special registers. However, whether or not such optimization is performed on an SQL statement or how often it is performed is not architected by DDM. For instance, this behavior may be a package attribute which is specified by a generic bind option.

**SEE ALSO**

None.

**NAME**

EXPYES — Explain All Explainable Static SQL Statements

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'240A'

**Length** \*

**Class** CODPNT

Explain All Explainable Static SQL Statements (EXPYES) specifies that the target SQLAM causes the target relational database (RDB) manager to explain normally any explainable static SQL statements during the bind or rebind process.

**SEE ALSO**

**insvar** *BNDEXPOPT* (on page 129)

**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)  
*SQLAM* (on page 847)

**NAME**

EXTDTA — Externalized FD:OCA Data

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'146C'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

EXTDTA consists of the externalized FD:OCA data corresponding to an FD:OCA placeholder in an FDOTA. (See *EXTDTAOVR* (on page 399) for more information).

An EXTDTA may be sent as command data (see *SQLDTA* (on page 860)), as part of the output data from processing an SQL statement (see *SQLDTARD* (on page 862)), or as part of query answer set data (QRYDTA). In the query case, the *qryblksz* parameter does not apply to the EXTDTA.

An EXTDTA object can optionally be generated in a streamed manner (see Layer B Streaming in *DSS* (on page 308) for details).

The externalized FD:OCA data objects in the EXTDTA carrier object must be encrypted with certain SECMECs. (See *EDTASECOVR* (on page 323) for details.)

<b>clsvar</b>	NIL
<b>insvar</b>	INSTANCE VARIABLES
<b>length</b>	*
<b>class</b>	X'146C'
<b>value</b>	INSTANCE_OF BYTSTRDR - Byte String REQUIRED
<b>clscmd</b>	NIL
<b>inscmd</b>	NIL

**SEE ALSO**

**insvar** CNTQRY (on page 222)  
 PRCCNVCD (on page 621)  
 RTNEXTDTA (on page 792)

**cmddta** EXCSQLSET (on page 377)  
 EXCSQLSTT (on page 381)

**rpydta** CNTQRY (on page 222)  
 EXCSQLSET (on page 377)  
 EXCSQLSTT (on page 381)  
 OPNQRY (on page 555)

**Semantic** CNTQRY (on page 222)  
 DHENC (on page 280)  
 EDTASECOVR (on page 323)  
 EUSRIDDTA (on page 355)  
 EUSRPWDDTA (on page 361)  
 EXCSQLSTT (on page 381)

*EXTDTAOVR* (on page 399)  
*FIXROWPRC* (on page 417)  
*LMTBLKPRC* (on page 475)  
*QRYBLK* (on page 670)  
*RTNEXTALL* (on page 791)  
*RTNEXTDTA* (on page 792)  
*RTNEXTROW* (on page 793)

**NAME**

EXTDTAOVR — Overview of Processing with Externalized FD:OCA Data

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Class** HELP

User FD:OCA data is transmitted between a source system and a target system as *base data*. This data is described by an FD:OCA descriptor that may flow with the data or independently of the data.

Base data flows in the following objects:

- SQLDTA, containing both the descriptor (FDODSC) and the data (FDODTA)
- SQLDTARD, containing both the descriptor (FDODSC) and the data (FDODTA)
- QRYDTA, containing only the data; the descriptor (QRYDSC) flows independently

Each such object is known as a *base data object*.

FD:OCA data consists of an ordered set of distinct data items, each of which may flow directly in the base data object or may flow as externalized data. If a data item is to flow as externalized data, the base data object contains an FD:OCA placeholder in place of the data item while the value bytes in the data item flow in an EXTDTA object. An EXTDTA must always contain value bytes. If a base column type is nullable, its EXTDTA object must have a one-byte null indicator that precedes any value bytes. As such, a nullable base data column that is null has no associated EXTDTA if this condition can be determined at the time its corresponding FD:OCA placeholder is generated. Also, an externalizable data column with a placeholder value of zero has no associated EXTDTA unless this condition cannot be determined until the time the EXTDTA is generated. Chapter 5 of the DRDA Reference provides additional details on when an EXTDTA object must be flowed.

An EXTDTA object can optionally be flowed in a streamed manner (see Layer B Streaming in *DSS* (on page 308) for details), in which case it must have a Layer B header length of X'8004'.

An unstreamed EXTDTA object containing a nullable value which is null has a Layer B header length of X'0005'.

An unstreamed EXTDTA object containing a nullable value which is not null but which has a length of zero bytes also has a Layer B header length of X'0005'.

An unstreamed EXTDTA object containing a non-nullable value which has a length of zero bytes has a Layer B header length of X'0004'.

See the following for a description of how EXTDTA objects flow in relation to the base data objects: *FIXROWPRC* (on page 417), *LMTBLKPRC* (on page 475), *EXTDTA* (on page 397), and *QRYBLK* (on page 670).

**SEE ALSO**

**Semantic** *EXTDTA* (on page 397)

## NAME

EXTENSIONS — Product Extensions to the DDM Architecture

## DESCRIPTION (Semantic)

**Dictionary** QDDTTRD**Length** \***Class** HELP

Product Extensions to the DDM Architecture (EXTENSIONS) enhance DDM file classes. The DDM file models and access methods are standardized descriptions (called classes) of file-oriented data management. These file classes cannot, however, stand alone. They require a supporting framework consisting of the following additional classes:

AGENT	Agent (see <i>AGENT</i> (on page 61))
CCSIDMGR	CCSID Manager (see <i>CCSIDMGR</i> (on page 150))
CMNMGR	Communications Manager (see <i>CMNMGR</i> (on page 196))
DICTIONARY	Dictionary (see <i>DICTIONARY</i> (on page 286))
SECMGR	Security Manager (see <i>SECMGR</i> (on page 814))
SUPERVISOR	Supervisor (see <i>SUPERVISOR</i> (on page 908))

Common data stream structures for the canonical representation of data objects, commands, and replies are also required.

This framework also supports product extensions to DDM. Some of these extensions pertain to multiple products and are, therefore, candidates for the development of "standardized" architecture classes. Other requirements are unique to single products, especially requirements for horizontal product growth or function distribution. While extensions are not candidates for standardization, the product's architectural definition is still required.

In both cases, the framework of existing DDM classes are used as the basis for extensions to DDM architecture. DDM allows the following "open architecture" enhancements:

- Whole new classes of objects (such as libraries or mailboxes) can be defined, either with new commands and replies unique to the class, or with existing DDM commands and replies as appropriate.
- Defining new commands for class can enhance the function of DDM classes.
- New parameters can be added to existing DDM commands.
- New values can be defined as valid for existing DDM parameters.

As new, standardized DDM architecture is developed, data stream codepoints are assigned to uniquely identify each structure. The DDM architecture assigns these codepoints and does not allow duplicate definitions. However, for product-defined extensions to DDM, products must be able to assign codepoints to their own structures without conflicting with the DDM architecture code points. The term CODPNTDR and the command EXCSAT accomplish just that.

Additional considerations for product extensions are described in SUBSETS.

**SEE ALSO**

**Semantic**

*CODPNTDR* (on page 235)  
*CONCEPTS* (on page 243)

## NAME

EXTNAM — External Name

## DESCRIPTION (Semantic)

**Dictionary** QDDBASD**Codepoint** X'115E'**Length** \***Class** CLASS**Sprcls** NAME - Name

The External Name (EXTNAM) is the name of the job, task, or process on a system for which a DDM server is active. On a source DDM server, the external name is the name of the job that is requesting access to remote resources. For a target DDM server, the external name is the name of the job the system creates or activates to run the DDM server.

No semantic meaning is assigned to external names in DDM.

External names are transmitted to aid in problem determination.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
length	*
class	X'115E'
value	INSTANCE_OF NAMDR - Name Data REQUIRED
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

## SEE ALSO

**insvar** *EXCSAT* (on page 363)  
*EXCSATRD* (on page 369)

**Semantic** *CCSIDMGR* (on page 150)



**NAME**

FALSE — False State

**DESCRIPTION (Semantic)****Dictionary** QDDPRMD**Codepoint** X'0018'**Length** \***Class** CONSTANT

The False State (FALSE) Boolean specifies the logical state of being false. This constant is also used wherever only one of two states can be specified.

**Literal Form**

The literal form of false is the capitalized FALSE.

---

value	X'F0'
-------	-------

**SEE ALSO**

**insvar**      *BOOLEAN* (on page 142)  
*FORGET* (on page 423)  
*OUTEXP* (on page 571)  
*QRYROWNBR* (on page 694)  
*RDBALWUPD* (on page 726)  
*RDBCMTOK* (on page 731)  
*RLSCONV* (on page 765)  
*RTNSQLDA* (on page 795)  
*SQLCSRHLD* (on page 858)  
*SYNCCTL* (on page 915)

**rpydta**      *PRPSQLSTT* (on page 636)

**Semantic**    *BOOLEAN* (on page 142)  
*CNTQRY* (on page 222)  
*OUTEXP* (on page 571)  
*QRYROWNBR* (on page 694)  
*RDBALWUPD* (on page 726)  
*RTNSQLDA* (on page 795)  
*SQLCSRHLD* (on page 858)  
*SYNCCTL* (on page 915)  
*SYNCPTOV* (on page 944)

**NAME**

FDOCA — Formatted Data Object Content Architecture

**DESCRIPTION (Semantic)**

**Dictionary** QDDTTRD

**Length** \*

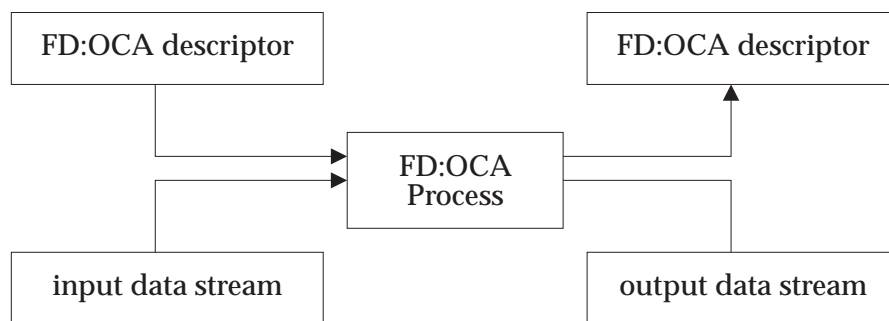
**Class** HELP

The Formatted Data Object Content Architecture (FD:OCA) is an architecture for describing, organizing, and manipulating a linear stream of data.

DDM uses FD:OCA primarily for the description of data for relational database (RDB) access. A complete description of FD:OCA, including how to build and interpret FD:OCA descriptors, is in the FD:OCA Reference.

The functions of FD:OCA are specified through an FD:OCA *descriptor* which consists of data structures called *triplets*. Attribute triplets describe the representation and layout of data in a data stream. Generator triplets specify how the data is manipulated to produce an output data stream.

An FD:OCA-containing architecture, such as DDM, transmits data streams and FD:OCA descriptors between communicating systems and invokes a presentation process as needed. The presentation process accepts both the data stream and the FD:OCA descriptor as input and produces a presentation stream as output, as shown in Figure 3-39. The FD:OCA-containing architecture processes any further presentation streams. For example, DDM can forward the presentation stream for storage to an RDB, or it can pass the presentation stream to an application requester.



**Figure 3-39** FD:OCA Processing Overview

**An Overview of Selected FD:OCA Triplets**

FD:OCA defines many more attribute and generator triplets than the current level of DDM architecture requires. This section provides an overview of the triplets.

**Scalar Data Arrays (SDA)**

SDAs are the triplets FD:OCA uses for describing data values that are either single items or linear or rectangular arrays of single items that all have the same format. DDM uses SDAs primarily to associate data representation specifications with DDM and SQL data types.

**Group Data Array (GDA)**

GDAs are triplets that define a group of data items as a referable unit. The elements of a GDA point (by label) to SDAs, other GDAs, or to RLOs to describe the data items of the group. The GDA can override certain attributes of each data representation.

**Row Layouts (RLO)**

RLOs are triplets that describe:

- A row containing fields of one or more types
- A table containing rows of one or more types
- Multi-dimensional, mixed data structures

RLOs describe data streams consisting of multiple unrelated structures. DDM uses RLOs primarily to describe the answer data that the SQL statements return.

**FD:OCA Descriptors**

An FD:OCA descriptor consists of one or more triplets laid out consecutively in a byte stream. Triplets that are referenced by other triplets must precede the referencing triplets. Unreferenced triplets are ignored.

**SEE ALSO**

None.

NAME

FDODSC — FD:OCA Data Descriptor

DESCRIPTION (Semantic)

**Dictionary** QDDPRMD  
**Codepoint** X'0010'  
**Length** \*  
**Class** CLASS  
**Sprcls** STRING - String

FD:OCA Data Descriptor (FDODSC) String is a DDM scalar whose value is a Formatted Data Object Content Architecture (FD:OCA) descriptor or a segment of an FD:OCA descriptor. An FDODSC consists of one or more FD:OCA triplets that describe the data fields contained in another scalar object.

See *FDOCA* (on page 404) for a description of the FD:OCA triplets DDM uses.

<b>clsvar</b>	NIL
<b>insvar</b>	CLASS INSTANCE VARIABLES
<b>length</b>	*
<b>class</b>	X'0010'
<b>value</b>	INSTANCE_OF BYTSTRDR - Byte String REQUIRED
<b>clscmd</b>	NIL
<b>inscmd</b>	NIL

SEE ALSO

**insvar** *DSCINVRM* (on page 298)  
*FDOOBJ* (on page 411)  
*SQLDTA* (on page 860)  
*SQLDTARD* (on page 862)  
*TYPDEF* (on page 1025)

**Semantic** *EXTDTAOVR* (on page 399)  
*FDODSCOFF* (on page 407)  
*FDODTA* (on page 408)  
*FDODTAOFF* (on page 409)  
*FDOEXT* (on page 410)  
*FDOOBJ* (on page 411)  
*FDOOFF* (on page 412)  
*FDOPRMOFF* (on page 413)  
*FDOTRPOFF* (on page 414)  
*QRYDSC* (on page 685)  
*TYPDEF* (on page 1025)

**NAME**

FDODSCOFF — FD:OCA Descriptor Offset

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2118'  
**Length** \*  
**Class** CLASS  
**Sprcls** BIN - Binary Integer Number

FD:OCA Descriptor Offset (FDODSCOFF) specifies the offset of the beginning byte of the Formatted Data Object Content Architecture (FD:OCA) descriptor in an FDODSC object when an FD:OCA error is being reported, and the complete descriptor is not returned. The offset is relative to the beginning of the complete Formatted Data Object Descriptor (FDODSC). A value of zero indicates that the FDODSC being returned begins with the first byte of the complete FDODSC.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	8	
class	X'2118'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	32
	MINVAL	0
	DFTVAL	0
	REQUIRED	
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *DSCINVRM* (on page 298)  
**Semantic** *DSCINVRM* (on page 298)

**NAME**

FDODTA — FD:OCA Data

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD  
**Codepoint** X'147A'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

FD:OCA Data (FDODTA) contains data that a Formatted Data Object Architecture descriptor (FDODSC) describes. The FDODSC may either be present with the Formatted Data Object Data (FDODTA) or may be implicitly defined based on the context of the command in which the FDODTA is used.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
<b>length</b>	*
<b>class</b>	X'147A'
<b>value</b>	INSTANCE_OF BYTSTRDR - Byte String REQUIRED
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO**

**insvar** *FDOOBJ* (on page 411)  
*SQLDTA* (on page 860)  
*SQLDTARD* (on page 862)

**Semantic** *EXTDTA* (on page 397)  
*EXTDTAOVR* (on page 399)  
*FDODTAOFF* (on page 409)  
*FDOEXT* (on page 410)  
*FDOOBJ* (on page 411)  
*FDOOFF* (on page 412)  
*QRYDTA* (on page 686)  
*SQLATTR* (on page 854)  
*SQLCARD* (on page 855)  
*SQLCINRD* (on page 857)  
*SQLDARD* (on page 859)  
*SQLOBINAM* (on page 866)  
*SQLRSLRD* (on page 867)  
*SQLSTT* (on page 868)  
*SQLSTTVRB* (on page 871)

**NAME**

FDODTAOFF — FD:OCA Data Offset

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2119'

**Length** \*

**Class** CLASS

**Sprcls** BIN - Binary Integer Number

FDOCA Data Offset (FDODTAOFF) specifies the offset of the beginning byte of the described data that appears in a Formatted Data Object Content Architecture data (FDODTA) object when an FD:OCA error is reported, and all of the data is not returned. The offset is relative to the beginning of the body of data sent that the FD:OCA descriptor (FDODSC) is interpreting. A value of zero indicates that the data being returned begins with the first byte of the data that was sent.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	8	
class	X'2119'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	32
	MINVAL	0
	DFTVAL	0
	REQUIRED	
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

None.

**NAME**

FDOEXT — FD:OCA Extent Data

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD  
**Codepoint** X'147B'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

FD:OCA Extent Data (FDOEXT) contains extent data for each SDA that a Formatted Data Object Architecture descriptor (FDODSC) describes. There is an FDOEXT entry for each field definition in the SQLDTAGRP in the FDODSC. The FDOEXT specification corresponding to a field definition in the FDODSC defines the number of times that field is repeated in the FDODTA object.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
<b>length</b>	<b>*</b>
<b>class</b>	<b>X'147B'</b>
<b>value</b>	<b>INSTANCE_OF BYTSTRDR - Byte String REQUIRED</b>
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO**

**insvar** *SQLDTA* (on page 860)



**NAME**

FDOOBJ — FD:OCA Object

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1480'

**Length** \*

**Class** CLASS

**Sprcls** ORDCOL - Ordered Collection

FD:OCA Object (FDOOBJ) is a self-describing data object consisting of an FD:OCA descriptor (FDODSC) and optionally an FD:OCA data object (FDODTA).

For multi-row inserts, a Formatted Data Content Architecture object (FDOOBJ) is a self-describing data object consisting of a Multi-Row Insert Data Descriptor (INSMRWDC) and a Multi-Row Insert Data (INSMRWDTA) object. The content of the FDOOBJ is based on the context of the command in which it is processed.

**DSS Carrier: OBJDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'1480'	
<b>fdodsc</b>	INSTANCE_OF REQUIRED REPEATABLE	FDODSC - FD:OCA Data Descriptor
<b>fdodta</b>	INSTANCE_OF OPTIONAL REPEATABLE	FDODTA - FD:OCA Data
<b>insmrwsc</b>	INSTANCE_OF REQUIRED REPEATABLE	INSMRWDC - Insert Data Descriptor
<b>insmrwdta</b>	INSTANCE_OF REQUIRED REPEATABLE	INSMRWDTA - Multi-Row Insert Data
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**Semantic** *SQLDTA* (on page 860)  
*SQLDTARD* (on page 862)

**NAME**

FDOOFF — FD:OCA Offset Data

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD  
**Codepoint** X'147D'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

FD:OCA Offset Data (FDOOFF) contains offset data for each SDA that a Formatted Data Object Architecture descriptor (FDODSC) describes. There is an FDOOFF entry for each field definition in the SQLDTAGRP in the FDODSC. The FDOOFF specification corresponding to a field definition in the FDODSC defines the offset to the start of the data entry in the FDODTA. The offset value for the first data array is zero (0).

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
<b>length</b>	*
<b>class</b>	X'147D'
<b>value</b>	INSTANCE_OF BYTSTRDR - Byte String REQUIRED
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO**

**insvar** *SQLDTA* (on page 860)

**NAME**

FDOPRMOFF — FD:OCA Triplet Parameter Offset

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'212B'

**Length** \*

**Class** CLASS

**Sprcls** BIN - Binary Integer Number

FD:OCA Triplet Parameter (FDOPRMOFF) specifies the offset to the byte of a Formatted Data Object Content Architecture (FD:OCA) triplet parameter that appears in an FD:OCA descriptor (FDODSC) object. The offset is relative to the first byte of an FD:OCA triplet. A value of zero indicates that the FD:OCA triplet parameter begins with the first byte of the FD:OCA triplet.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	6	
class	X'212B'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	16
	MINVAL	0
	REQUIRED	
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *DSCINVRM* (on page 298)

**NAME**

FDOTRPOFF — FD:OCA Triplet Offset

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'212A'  
**Length** \*  
**Class** CLASS  
**Sprcls** BIN - Binary Integer Number

FD:OCA Triplet Offset (FDOTRPOFF) Binary Integer Number specifies the offset to the first byte of a Formatted Data Object Content Architecture (FD:OCA) triplet that appears in an FD:OCA descriptor (FDODSC) object. The offset is relative to the first value byte of the FDODSC object. A value of zero indicates that the FD:OCA triplet begins with the first value byte of the FDODSC object.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
<b>length</b>	8	
<b>class</b>	X'212A'	
<b>value</b>	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	32
	MINVAL	0
	REQUIRED	
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *DSCINVRM* (on page 298)  
**Semantic** *DSCINVRM* (on page 298)

## NAME

FIELD — A Discrete Unit of Data

## DESCRIPTION (Semantic)

**Dictionary** QDDPRMD**Codepoint** X'006A'**Length** \***Class** CLASS**Sprcls** DATA - Encoded Information

A Discrete Unit of Data (FIELD) is data that in general does not describe its own length or format. The association of a field with its class is through an object, such as a record format or the class description of a scalar. There is no command protocol for fields.

Each subclass of FIELD specifies the attributes that are specified in describing its instances and specifies the valid lengths or range of lengths of its instances.

Class FIELD is only used as an abstract superclass to represent the concept of discrete units of data.

<b>clsvar</b>	<b>CLASS VARIABLES</b>	
<b>vldattls</b>	INSTANCE_OF TITLE NOTE	DEFLST - Definition List  Valid field attributes. Specifies the list of attributes that can be specified for a given subclass of FIELD.
<b>insvar</b>	NIL	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	
<b>vldattls</b>	NIL	

## SEE ALSO

**clsvar** *ENUVAL* (on page 347)

**insvar!T{** *CNSVAL* (on page 221)  
*DFTVAL* (on page 276)  
*MAXVAL* (on page 499)  
*MINVAL* (on page 519)  
*RESERVED* (on page 762)  
*SPCVAL* (on page 831)

**Semantic** *BINDR* (on page 120)  
*BITDR* (on page 123)  
*BITSTRDR* (on page 124)  
*BYTDR* (on page 145)  
*BYTSTRDR* (on page 146)  
*CHRDR* (on page 155)  
*CHRSTRDR* (on page 156)  
*CODPNTDR* (on page 235)  
*HEXDR* (on page 428)  
*HEXSTRDR* (on page 429)

*NAMDR* (on page 526)  
*NAMSYMDR* (on page 528)  
*OBJECT* (on page 540)  
*PKGCNSTKN* (on page 588)  
*PKGID* (on page 591)  
*PKGSN* (on page 606)  
*RDBCOLID* (on page 732)  
*TIMEOUT* (on page 1019)  
*XAFLAGS* (on page 1061)

**NAME**

FIXROWPRC — Fixed Row Query Protocol

Name of term prior to Level 4: SNGROWPRC

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2418'

**Length** \*

**Class** CODPNT

Fixed Row Query Protocol (FIXROWPRC) supports both non-scrollable and scrollable cursors and is required for rowset cursors in support of multi-row fetch. For more information on scrollable cursors, see the DRDA Reference, Appendix B.

Fixed Row Query Protocol is used for queries that are declared with rowset positioning and for the return of result sets declared with rowset positioning by the invocation of a stored procedure. Refer to *qryattset* for more details on rowset processing.

The fixed row query protocols are used for non-rowset queries that have an ambiguous cursor if:

- The OPNQRY command specified FRCFIXROW in the QRYBLKCTL parameter.
- The package QRYBLKCTL parameter value is either FIXROWPRC or FRCFIXROW.

The fixed row query protocols are used for non-rowset queries that have a known database cursor if:

- The database cursor is either declared for UPDATE or may be the target of a WHERE CURRENT OF clause on an SQL UPDATE or DELETE statement.
- The OPNQRY command specified FRCFIXROW in the QRYBLKCTL parameter.
- The package QRYBLKCTL parameter is FRCFIXROW.

The fixed row query protocols are also used if the cursor is scrollable (QRYATTSCR is TRUE) and has sensitivity of sensitive dynamic (QRYATTSNS is QRYSNSDYN).

The fixed row query protocols are also used if the scrollable or non-scrollable cursor returns LOB data and the OPNQRY command specified an OUTOVROPT of OUTOVRANY.

If the cursor is scrollable, the application requester may either use the cursor in a scrollable or non-scrollable manner as follows (see the DRDA Reference, Appendix B for more details):

- For application requesters that wish to provide support for scrollable, non-rowset cursors, a QRYROWSET value may be provided on subsequent CNTQRY commands for non-dynamic scrollable cursors. This allows more than one single-row FETCH to be requested by the target SQLAM and for the rows retrieved to be returned as members of a DRDA rowset. If a QRYROWSET value is specified, the application requester is responsible for managing cursor position differences that may result. If a QRYROWSET value is not specified, this is not required. QRYROWSET is not allowed for dynamic scrollable cursors (QRYATTSNS is QRYSNSDYN) since it is not possible to manage cursor positions by row number.
- For application requesters that do not wish to support scrollable cursors, the scrollable cursor can be accessed in a non-scrollable fashion with subsequent CNTQRY commands that do not specify any scrolling parameters.

The OPNQRY command initiates the query process. After each OPNQRY (or CNTQRY) command, the query is suspended unless the query is terminated by an error condition (see

*OPNQRY* (on page 555)). The *CNTQRY* command resumes a suspended query so that the next portion of answer set data can be returned. A query is terminated anytime the *CLSQRY* command suspends it.

The normal response to an *OPNQRY* command is:

- An *OPNQRYRM* is returned.
- A description of the query answer set data (see *QRYDSC* (on page 685) and the DRDA Block Format rules (BF) in the DRDA Reference for detailed information).
- Possibly answer set data is returned as one or more *QRYDTA* reply data objects following the query description (see the DRDA Query Data Transfer Protocols rules (QP) in the DRDA Reference for detailed information).

For non-scrollable cursors, no *QRYDTA* data objects are returned in response to the *OPNQRY* command.

For scrollable cursors, the use of the *QRYROWSET* parameter may result in the return of one or more *QRYDTA* objects. If a *QRYROWSET* value is specified and honored, the application requester is responsible for managing cursor position differences that may result (see the DRDA Reference, Appendix B for more details).

If the answer set includes any LOB data whose output form must be resolved at the target when the data is fetched (LOB data *versus* LOB locator form), no *QRYDTA* is returned until the first *CNTQRY* command is received.

- If answer set data is returned, possibly, additional query blocks, limited in number by the value of the *MAXBLKEXT* parameter of *OPNQRY*, each containing a *QRYDTA* reply data object, are returned.
- The query or result set is suspended.

The normal response to a *CNTQRY* command is:

- One or more *QRYDTA* reply data objects are returned.
- The query or result set is suspended.

The target *SQLAM* must use the block size specified in the *OPNQRY* or *CNTQRY* commands. The way in which this value is used by the target *SQLAM* to build the *QRYDSC* and *QRYDTA* objects depends on the *SQLAM* level. See the DRDA Blocking Rules (BF and BS) in the DRDA Reference for detailed information.

Each *QRYDTA* reply data object contains data from one or more answer set rows with associated *SQLCAs*. In the case of flexible query blocks, a *QRYDTA* can contain only whole rows. In the case of exact query blocks, a *QRYDTA* may contain only a portion of a row. The target *SQLAM* must use the block size specified on the *OPNQRY* or *CNTQRY* commands.

In the following sections, everything that appears on a line preceded by `<==` represents a single object, and `(--)` represents the parameter list.

*TYPDEFNAM* and *TYPDEFOVR* reply data objects might precede an *SQLCARD* and *QRYDSC* reply data objects to override the *CCSIDs*. One example of *TYPDEFNAM* and *TYPDEFOVR* in each of the following protocols is shown.

If there are LOBs in the output, the following applies.

The application server must select this query protocol if the query returns any LOB data columns and the application requester has specified that output overrides may be sent with each *CNTQRY* command.



The normal response to a CNTQRY command is:

- Query Answer set data is returned, in one of the following ways:
  - One or more QRYDTA reply data objects are returned.
  - One or more QRYDTA reply data objects and one or more EXTDTA reply data objects are returned.
- The query is suspended.

### Protocols for the OPNQRY Command

The following examples show some of the valid responses to the OPNQRY command:

- For normal and non-terminating (warning) error conditions, an OPNQRYRM is returned followed a QRYDSC reply data object that contains the description of the answer set data.

```
OPNQRY (--) ==>
                <== OPNQRYRM (--) QRYDSC (--)
                --- or ---
```

```
OPNQRY (--) ==>
                <== OPNQRYRM (--) SQLCARD (--) QRYDSC (--)
                --- or ---
```

```
OPNQRY (--) ==>
                <== OPNQRYRM (--) TYPDEFNAM (--) TYPDEFOVR (--)
                SQLCARD (--) QRYDSC (--)
```

If the cursor is scrollable and the application requester specified a non-zero QRYROWSET parameter on OPNQRY command, then the application server may return one or more QRYDTA objects as in response to an OPNQRY for the FIXROWPRC protocol.

- When an OPNQRY is issued for a query that is currently suspended (opened by a previous OPNQRY command and is not terminated), a QRYPOPRM is returned. The query remains suspended.

```
OPNQRY (--) ==>
                <== QRYPOPRM (--)
```

- When the target RDB detects an error condition preventing the database cursor from being opened prior to the OPNQRYRM being returned, an OPNQFLRM reply message is returned followed by an SQLCARD object. This terminates the query.

```
OPNQRY (--) ==>
                <== OPNQFLRM (--) SQLCARD (--)
```

- For terminating error conditions, a reply message is returned. This causes the query to be terminated. The reply message precedes an SQLCARD object.

```
OPNQRY (--) ==>
                <== ABNUOWRM (--) SQLCARD (--)
```

### Protocols for the CNTQRY Command

The following examples show some of the valid responses to the CNTQRY command:

- For normal and non-terminating error conditions:
  - For a non-rowset cursor, if only a single row is requested, then one or more QRYDTA reply data objects are returned that contain an SQLCA and a single row of answer data for single row fetches. If a DRDA rowset was requested by means of the QRYROWSET parameter, then more than one row may be returned in one or more QRYDTA objects. When the server is using flexible query blocks, a row is completely contained in one QRYDTA object. When the server is using exact query blocks, a row may span multiple QRYDTA objects.
  - For a rowset cursor, one or more QRYDTA reply data objects are returned that contain the multiple row answer rows with associated SQLCAs. When the server is using flexible query blocks, an SQL rowset is completely contained in one QRYDTA object.

If any row in the answer set contains data that will be flowed as externalized FD:OCA data in a DDM EXTDTA, then the EXTDTA objects associated with a row flow after the QRYDTA object that contains the complete row.

```
CNTQRY (--) ==>
                <== QRYDTA (--)
                --- or ---
```

```
CNTQRY (--) ==>
                <== QRYDTA (--)
                <== QRYDTA (--)
                <== QRYDTA (--)
                --- or ---
```

```
CNTQRY (--) ==>
                <== QRYDTA (--)
                <== EXTDTA (--)
                <== EXTDTA (--)
                <== EXTDTA (--)
```

- For non-scrollable cursors, if a previous CNTQRY command returns the last row of answer set data (SQLSTATE 02000), and the query is not terminated, subject to the cursor type and the QRYCLSIMP value as specified previously on the OPNQRY command, an ENDQRYRM and an SQLCARD object may be returned. This terminates the query.
- For multi-row fetches, if the target SQLAM is aware that the last row of answer set data is not being returned, then an SQLCARD object is placed following the QRYDTA reply data object.

```
CNTQRY (--) ==>
                <== QRYDTA (--) SQLCARD (--)
                --- or ---
```

```
CNTQRY (--) ==>
                <== QRYDTA (--) EXTDTA (--) SQLCARD (--)
```

- For multi-row fetches, if the target SQLAM is aware that the last row of answer set data is being returned, then an ENDQRYRM and an SQLCARD object are placed following the QRYDTA reply data object. This terminates the query.

```

CNTQRY (--) ==>
                <== QRYDTA (--) ENDQRYRM (--) SQLCARD (--)
                --- or ---
CNTQRY (--) ==>
                <== QRYDTA (--) EXTDTA (--) ENDQRYRM (--) SQLCARD (--)

```

- When an error condition results in query termination, a reply message is returned and an SQLCARD object follows. This causes the query to terminate. The reply message is always followed by an SQLCARD object.

```

CNTQRY (--) ==>
                <== ENDQRYRM (--) TYPDEFNAM (--)
                TYPDEFOVR (--) SQLCARD (--)

```

### Protocols for the CLSQRY Command

The following examples show some of the valid responses to the CLSQRY command.

- For normal and non-terminating error conditions detected when the query is suspended, an SQLCARD object is returned. A CLSQRY never returns an ENDQRYRM. The query is terminated.

```

CLSQRY (--) ==>
                <== SQLCARD (--)
                --- or ---
CLSQRY (--) ==>
                <== TYPDEFNAM (--) TYPDEFOVR (--) SQLCARD (--)

```

- When the query is terminated or not open, the QRYNOPRM reply message is returned indicating the query is not open.

```

CLSQRY (--) ==>
                <== QRYNOPRM (--)

```

- For terminating error conditions that prevent the CLSQRY command from being passed to the target SQLAM, an appropriate reply message is returned.

```

CLSQRY (--) ==>
                <== RDBNACRM (--)

```

### SEE ALSO

<b>clsvar</b>	<i>QRYPRCTYP</i> (on page 692)
<b>insvar</b>	<i>CNTQRY</i> (on page 222) <i>QRYBLKCTL</i> (on page 672)
<b>Semantic</b>	<i>CODPNT</i> (on page 233) <i>CNTQRY</i> (on page 222) <i>ENDQRYRM</i> (on page 342) <i>EXTDIAOVR</i> (on page 399) <i>EXTDIAOVR</i> (on page 399) <i>EXCSQLSTT</i> (on page 381) <i>OPNQRY</i> (on page 555) <i>OUTOVRANY</i> (on page 574) <i>QDDRDBD</i> (on page 657)

*QRYBLK* (on page 670)  
*QRYBLKCTL* (on page 672)  
*QRYCLSIMP* (on page 680)  
*QRYROWSET* (on page 697)  
*SQLAM* (on page 847)

**NAME**

FORGET — Forget Unit of Work

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD  
**Codepoint** X'1186'  
**Length** \*  
**Class** CLASS  
**Sprcls** BOOLEAN - Logical Value

An explicit forget (FORGET) is requested by the source sync point manager in order to forget all knowledge of the unit of work. When the sync control committed request is sent, the source sync point manager identifies the type of forget processing to be performed by the target sync point manager. If FORGET is set to TRUE, the target sync point manager responds with the SYNCCRD forget reply; otherwise, the forget is implied by the next response from the target server. If the conversation is to be terminated after the sync point operation is completed if the unit of work is committed, forget must be set to TRUE.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	5	
class	X'1186'	
value	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	LENGTH	2
	ENUVAL	X'F1' - TRUE - True State
	NOTE	X'F1' - SYNCCRD SYNCTYPE(FORGET) requested.
	ENUVAL	X'F0' - FALSE - False State
	NOTE	X'F0' - Next reply implies forget.
	DFTVAL	X'F0' - FALSE - False State
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *RSYNCTYP* (on page 790)  
*SYNCCTL* (on page 915)  
**Semantic** *SYNCCRD* (on page 981)  
*SYNCRSY* (on page 982)  
*XAMGR* (on page 1063)

**NAME**

FRCFIXROW — Force Fixed Row Query Protocol

Name of term prior to Level 4: FRCSNGROW

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2410'

**Length** \*

**Class** codpnt

The Force Fixed Row Query Protocol (FRCFIXROW) codepoint specifies the fixed row query protocol to be used. If the BGNBND command specifies FRCFIXROW, then the fixed row query protocol must be used for all database cursors. If OPNQRY specifies FRCFIXROW, then the fixed row protocol is used only for the current database cursor.

**SEE ALSO**

**insvar** *OPNQRY* (on page 555)  
*QRYBLKCTL* (on page 672)

**Semantic** *FIXROWPRC* (on page 417)  
*LMTBLKPRC* (on page 475)  
*QRYBLKCTL* (on page 672)

**NAME**

HELP — Help Text

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD  
**Codepoint** X'0019'  
**Length** \*  
**Class** CLASS  
**Sprcls** DATA - Encoded Information

Help Text (HELP) provides extended information about a topic.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'0019'	
status	INSTANCE_OF TITLE	STSLST - Term Status Collection term status
title	INSTANCE_OF	TITLE - Title
<b>semantic</b>	INSTANCE_OF MAXLEN REPEATABLE TITLE	TEXT - Text Character String 80 For each line of help text. description
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**Semantic** *AGNCMDPR* (on page 64)  
*AGNRPYPR* (on page 67)  
*APPCMNFL* (on page 68)  
*APPCMNI* (on page 72)  
*APPCMNT* (on page 76)  
*APPSRCCD* (on page 79)  
*APPSRCCR* (on page 86)  
*APPSRCER* (on page 91)  
*APPTRGER* (on page 95)  
*CLASS* (on page 158)  
*CMNLYR* (on page 194)  
*CMNOVR* (on page 201)  
*CODPNT* (on page 233)  
*CONCEPTS* (on page 243)  
*CONSTANT* (on page 244)  
*DCESECOVR* (on page 253)  
*DCESECTKN* (on page 257)  
*DDM* (on page 260)  
*DDMOBJ* (on page 262)  
*DHENC* (on page 280)  
*DICTIONARY* (on page 286)  
*DTAOVR* (on page 321)

*EDTASECOVR* (on page 323)  
*EXTDTAOVR* (on page 399)  
*EXTENSIONS* (on page 400)  
*FDOCA* (on page 404)  
*INHERITANCE* (on page 437)  
*LVLCMP* (on page 486)  
*MGROVR* (on page 514)  
*NWPWDSEC* (on page 535)  
*OBJOVR* (on page 544)  
*OOPOVR* (on page 547)  
*OSFDCE* (on page 570)  
*PLGINSECTKN* (on page 620)  
*PRCOVR* (on page 628)  
*PWDENC* (on page 644)  
*PWDSBS* (on page 646)  
*PWDSEC* (on page 649)  
*QRYBLK* (on page 670)  
*RDBOVR* (on page 740)  
*RESYNOVR* (on page 764)  
*SECOVR* (on page 819)  
*SNASECOVR* (on page 828)  
*SQL* (on page 835)  
*SRVOVR* (on page 881)  
*STRLYR* (on page 890)  
*SUBSETS* (on page 902)  
*SYNCMNBK* (on page 924)  
*SYNCMNCM* (on page 926)  
*SYNCMNFL* (on page 928)  
*SYNCMNI* (on page 931)  
*SYNCMNT* (on page 935)  
*SYNCPTOV* (on page 944)  
*TASK* (on page 991)  
*TCPCMNFL* (on page 992)  
*TCPCMNI* (on page 994)  
*TCPCMNT* (on page 997)  
*TCPIPOVR* (on page 1000)  
*TCPSRCCD* (on page 1007)  
*TCPSRCCR* (on page 1010)  
*TCPSRCER* (on page 1013)  
*TCPTRGER* (on page 1015)  
*USRIDSEC* (on page 1048)  
*UNORDERED* (on page 1036)  
*USRSECOVR* (on page 1050)  
*XAMGROV* (on page 1066)

**vldattls**

*AGENT* (on page 61)  
*CCSIDMGR* (on page 150)  
*CMNAPPC* (on page 184)  
*CMNMGR* (on page 196)  
*CMNSYNCPT* (on page 202)  
*CMNTCPIP* (on page 214)  
*DICTIONARY* (on page 286)  
*MANAGER* (on page 491)



*RDB* (on page 718)  
*RSYNCMGR* (on page 787)  
*SECMGR* (on page 814)  
*SERVER* (on page 824)  
*SQLAM* (on page 847)  
*SUPERVISOR* (on page 908)  
*SYNCPTMGR* (on page 939)

**NAME**

HEXDR — Hexadecimal Number  
 Name of term prior to Level 2: HEX

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD  
**Codepoint** X'001A'  
**Length** \*  
**Class** CLASS  
**Sprcls** FIELD - A Discrete Unit of Data

Hexadecimal Number (HEXDR) Field specifies a four-bit encoding. Only one of sixteen values can be assigned.

**Length Specification**

The length of a HEXDR field is 1 nibble (a nibble is 4 bits).

**Literal Form**

Hex literals are specified in the form X'h' where h is a character in the string '0123456789ABCDEF'.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
value	INSTANCE_OF LENGTH REQUIRED	BITSTRDR - Bit String Field 4
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	
<b>vldattls</b>	NIL	

**SEE ALSO**

**clsvar** *ENUVAL* (on page 347)  
**insvar** *CNSVAL* (on page 221)  
*CODPNTDR* (on page 235)  
*DFTVAL* (on page 276)  
*DSSFMT* (on page 318)  
*HEXSTRDR* (on page 429)  
*MAXVAL* (on page 499)  
*MINVAL* (on page 519)  
*RESERVED* (on page 762)  
*SPCVAL* (on page 831)  
**Semantic** *HEXSTRDR* (on page 429)  
*INHERITANCE* (on page 437)

**NAME**

HEXSTRDR — Hexadecimal String  
 Name of term prior to Level 2: HEXSTR

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD  
**Codepoint** X'001B'  
**Length** \*  
**Class** CLASS  
**Sprcls** FIELD - A Discrete Unit of Data

Hexadecimal String (HEXSTRDR) Field specifies a string of hexadecimal numbers.

**Length Specification**

The length of a HEXSTRDR field is expressed in units of HEXDR.

**Literal Form**

Hex string literals are specified in the form X'h...' where *h* is a character in the string '0123456789ABCDEF'.

<b>clsvar</b>	NIL
<b>insvar</b>	CLASS INSTANCE VARIABLES
value	INSTANCE_OF REPEATABLE REQUIRED HEXDR - Hexadecimal Number
<b>clscmd</b>	NIL
<b>inscmd</b>	NIL
<b>vldattls</b>	NIL

**SEE ALSO**

- clsvar** *ENUVAL* (on page 347)
- insvar** *BOOLEAN* (on page 142)  
*CCSIDDBC* (on page 148)  
*CCSIDMBC* (on page 149)  
*CCSIDSBC* (on page 154)  
*CMMTYP* (on page 183)  
*CNSVAL* (on page 221)  
*CODPNTDR* (on page 235)  
*DEPERRCD* (on page 270)  
*DFTVAL* (on page 276)  
*DIAGLVL* (on page 284)  
*FORGET* (on page 423)  
*MAXVAL* (on page 499)  
*MINVAL* (on page 519)  
*OUTEXP* (on page 571)  
*PRCCNVCD* (on page 621)  
*QRYATTSET* (on page 666)

*QRYCLSIMP* (on page 680)  
*QRYCLSRLS* (on page 682)  
*RDBALWUPD* (on page 726)  
*RDBCMTOK* (on page 731)  
*RESERVED* (on page 762)  
*RLSCONV* (on page 765)  
*RTNSETSTT* (on page 794)  
*RTNSQLDA* (on page 795)  
*SECCHKCD* (on page 806)  
*SPCVL* (on page 831)  
*SQLCSRHLD* (on page 858)  
*SYNERRCD* (on page 985)  
*UOWDSP* (on page 1037)  
*XAFLAGS* (on page 1061)

**Semantic**

*DIAGLVL* (on page 284)  
*DSSFMT* (on page 318)  
*DUPQRYOK* (on page 322)  
*INHERITANCE* (on page 437)

**NAME**

HMSBLKTIMFMT — HMS with Blank Separator Time Format

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'240C'**Length** \***Class** CODPNT

The HMS (Hours, Minutes, Seconds) with Blank Separator Time Format (HMSBLKTIMFMT) specifies that times in the Structured Query Language (SQL) statements are in the following HMS time format:

hh mm ss

where the " " separator is a blank (with the Graphic Character Global Identifier (GCGID) SP01). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO****insvar** *STTTIMFMT* (on page 900)**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)

**NAME**

HMSCLNTIMFMT — HMS with Colon Separator Time Format

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'240D'**Length** \***Class** CODPNT

The HMS (Hours, Minutes, Seconds) with Colon Separator Time Format (HMSCLNTIMFMT) specifies that times in the Structured Query Language (SQL) statements are in the following HMS time format:

hh:mm:ss

where the ":" separator is a colon (with the Graphic Character Global Identifier (GCGID) SP13). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

HMSCLNTIMFMT shares a common format with JISTIMFMT. They are kept separate for reporting purposes.

**SEE ALSO****insvar** *STTIMFMT* (on page 900)**Semantic** *CODPNT* (on page 233)  
*JISTIMFMT* (on page 459)  
*QDDRDBD* (on page 657)

**NAME**

HMSCMATIMFMT — HMS with Comma Separator Time Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2416'

**Length** \*

**Class** CODPNT

The HMS (Hours, Minutes, Seconds) with Comma Separator Time Format (HMSCMATIMFMT) specifies that times in the Structured Query Language (SQL) statements are in the following HMS time format:

hh, mm, ss

where the "," separator is a comma (with the Graphic Character Global Identifier (GCGID) SP08). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar** *STTTIMFMT* (on page 900)

**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)

**NAME**

HMSPRDTIMFMT — HMS with Period Separator Time Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2428'

**Length** \*

**Class** CODPNT

The HMS (Hours, Minutes, Seconds) with Period Separator Time Format (HMSPRDTIMFMT) specifies that times in the Structured Query Language (SQL) statements are in the following HMS time format:

hh.mm.ss

where the "." separator is a comma (with the Graphic Character Global Identifier (GCGID) SP11). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

HMSPRDTIMFMT shares a common format with ISOTIMFMT and EURTIMFMT. They are kept separate for reporting purposes.

**SEE ALSO**

**insvar** *STTIMFMT* (on page 900)

**Semantic** *CODPNT* (on page 233)  
*EURTIMFMT* (on page 354)  
*ISOTIMFMT* (on page 457)  
*QDDRDBD* (on page 657)



**NAME**

IGNORABLE — Ignorable Value Attribute

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'001C'

**Length** \*

**Class** CLASS

**Sprcls** OBJECT - Self-identifying Data

Ignorable Value Attribute (IGNORABLE) specifies that the receiver can ignore a parameter of a command or the value of a parameter if the receiver does not provide the support requested.

All senders optionally send the parameter or value.

All receivers must recognize the parameter or value.

The receiver is not required to support the architected default value or validate the specified value.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
<b>length</b>	<b>*</b>
<b>class</b>	<b>X'001C'</b>
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO**

- insvar** *ACCRDB* (on page 42)
- ACCSEC* (on page 52)
- BGNBND* (on page 110)
- EXCSAT* (on page 363)
- REBIND* (on page 753)
- SECCHK* (on page 800)
- SECCHKRM* (on page 809)

**NAME**

INFO — Information Only Severity Code

**DESCRIPTION (Semantic)****Dictionary** QDDPRMD**Codepoint** X'001E'**Length** \***Class** CONSTANT

Information Only Severity Code (INFO) specifies that a reply message only contains information, not a description of any *problem*.

---

value	0
-------	---

**SEE ALSO**

<b>insvar</b>	<i>ACCRDBRM</i> (on page 48) <i>CMDCHKRM</i> (on page 173) <i>CMDCMPRM</i> (on page 175) <i>OPNQRYSRM</i> (on page 566) <i>RDBUPDRM</i> (on page 751) <i>RSLSETRM</i> (on page 785) <i>SECCHKRM</i> (on page 809) <i>SVRCOD</i> (on page 911)
<b>Semantic</b>	<i>SECCHK</i> (on page 800) <i>SECCHKCD</i> (on page 806) <i>USRSECOVR</i> (on page 1050)

**NAME**

INHERITANCE — Class Inheritance

**DESCRIPTION (Semantic)**

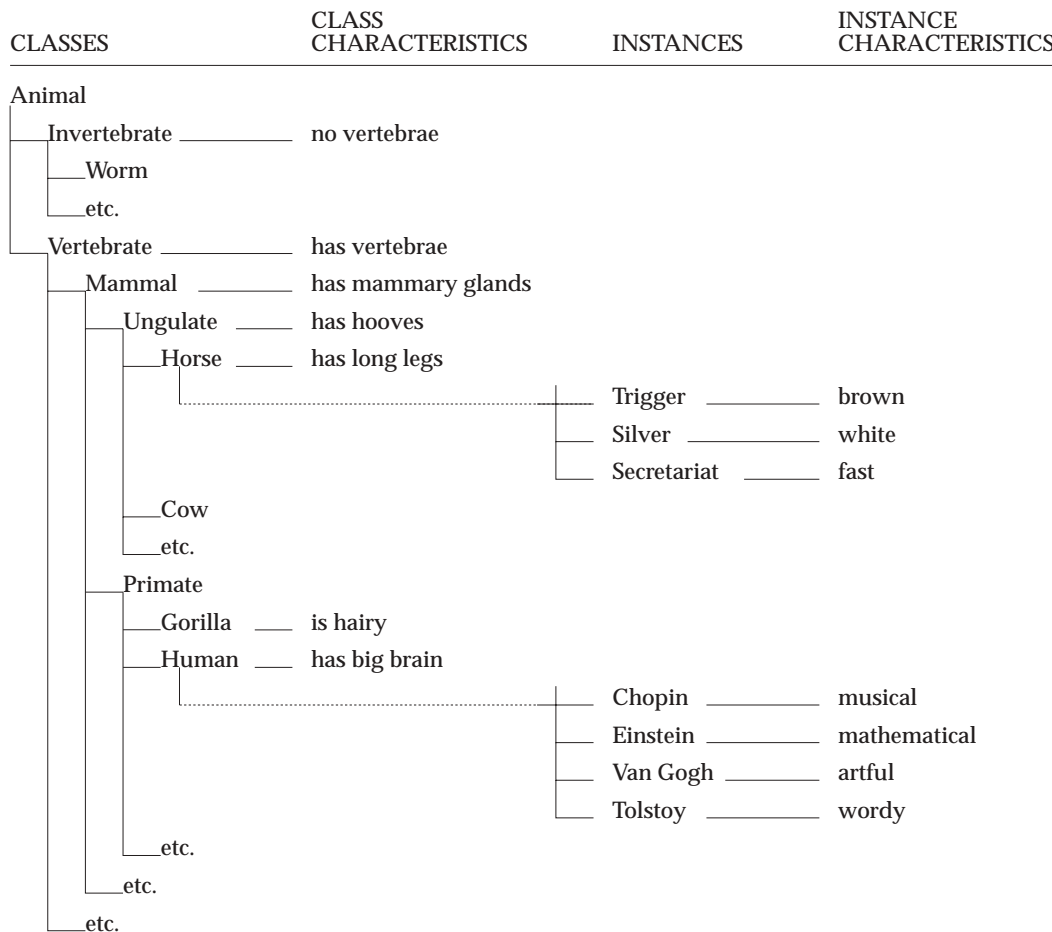
**Dictionary** QDDTTRD

**Length** \*

**Class** HELP

The Class Inheritance (INHERITANCE) is a means of ensuring consistency and accuracy within DDM architecture.

An analogy of inheritance is illustrated in Figure 3-40 which displays the inheritance of a zoological taxonomy.



**Figure 3-40** Inheritance in a Zoological Taxonomy

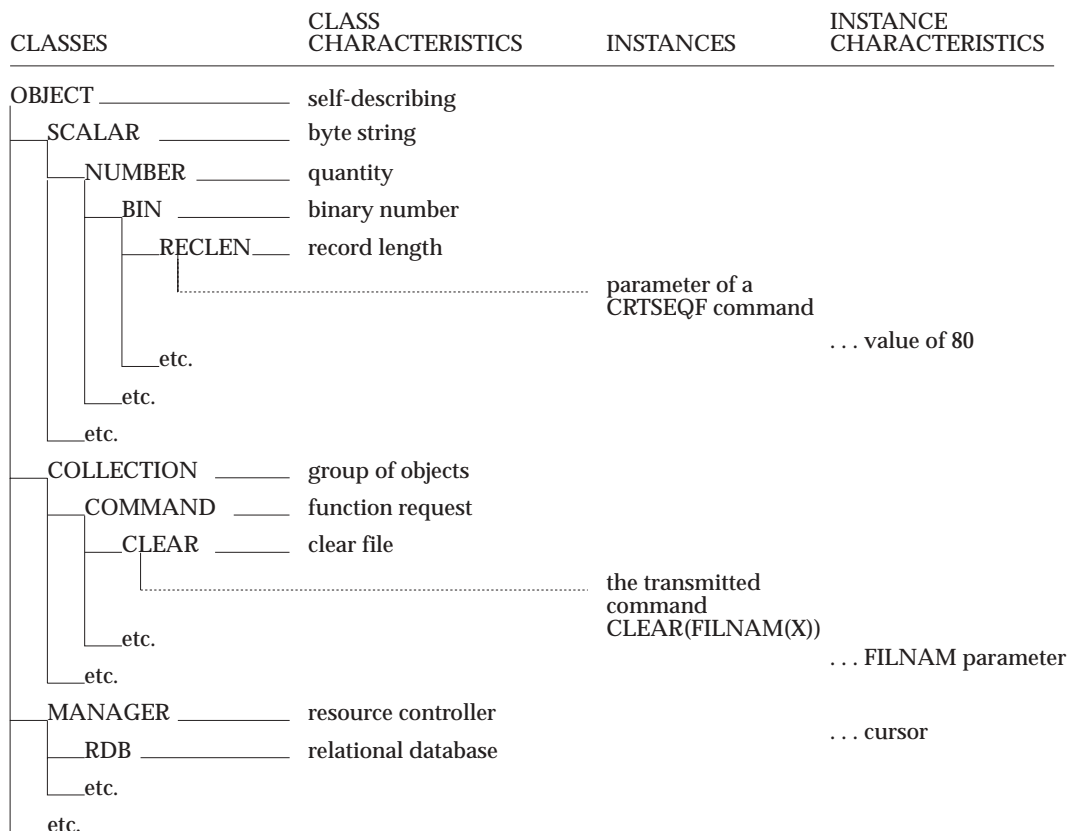
The class of Horse has characteristics unique to horses, but it also inherits characteristics from the classes of ungulates, mammals, vertebrates, and animals. That is, horses have vertebrae, mammary glands, and hooves in addition to having long legs.

The class Mammal is a superclass of class Horse and a subclass of class Vertebrate. Particular horses, such as Silver and Secretariat, as instances of class horse, inherit all of the direct and inherited characteristics of horses while exhibiting unique characteristics. Both classes and

instances provide an easy reference; for instance, mammal, vertebrate, Chopin, and Secretariat.

Horses are referred to as animals while at the same time a general discussion of animals includes all horses.

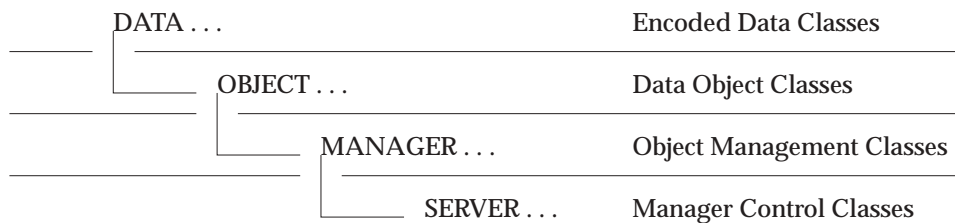
In DDM architecture, these same techniques are applied to the domain of data as illustrated in Figure 3-41. Classes of data are organized in a hierarchy of inherited structures and characteristics. Instances of these classes are created as needed for particular applications. Both classes and instances provide an easy reference.



**Figure 3-41** Inheritance in the DDM Taxonomy

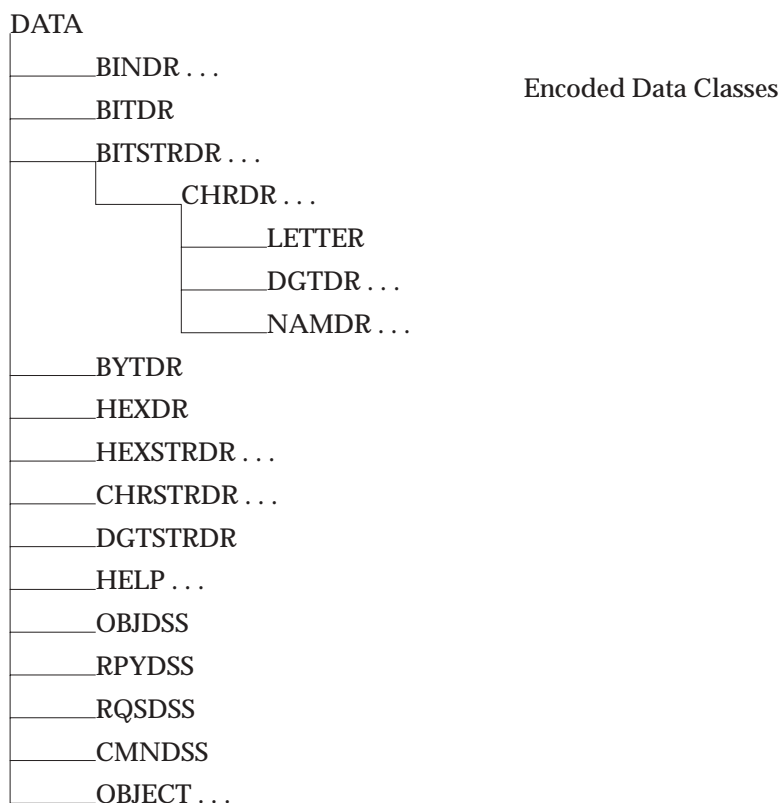
A file is referred to as an object while at the same time a general discussion of objects includes all files.

The overall DDM hierarchy is divided into levels in Figure 3-42 (on page 439) to provide an overview. Each level is more fully illustrated in the subsequent figures. Ellipses (...) indicate that additional subclasses of the term exist. Only the basic hierarchy of DDM is shown. The terms named are the superclasses of the terms that define the DDM architecture.



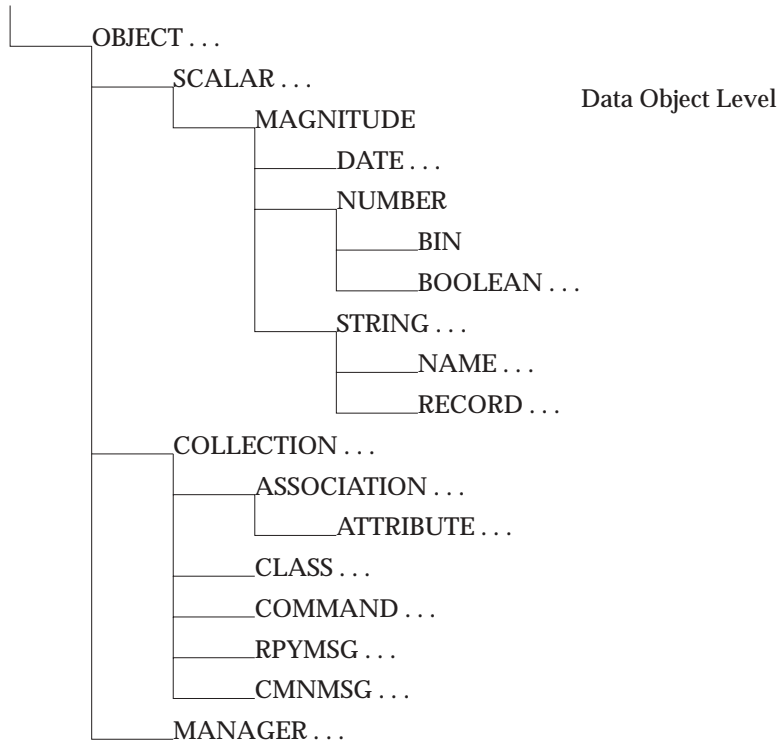
**Figure 3-42** Levels of the DDM Descriptive Hierarchy

The encoded data classes of DDM represent information as a pattern of bits in memory or in a message. The specific subclasses of the DATA class in Figure 3-43 define the encoding scheme.



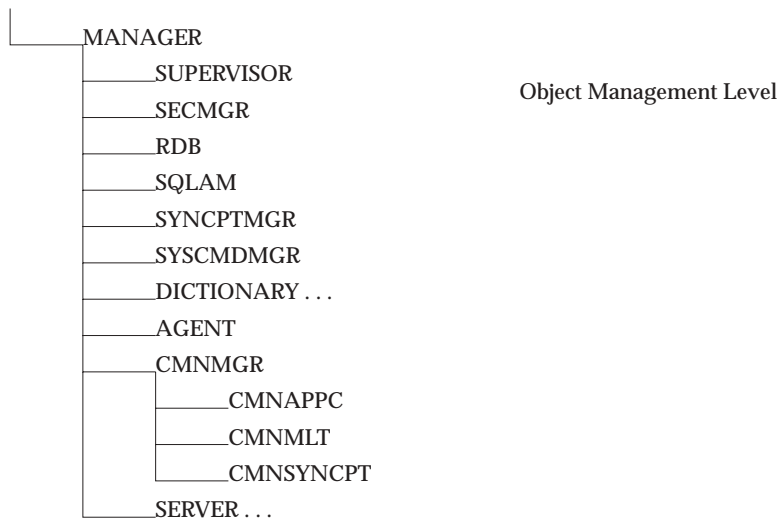
**Figure 3-43** Encoded DATA Class Hierarchy

The data object classes of DDM are self-defining structures that identify a data object's length and class (see Figure 3-44 (on page 440)). The value of a SCALAR object is some form of encoded data as the specific subclass of the SCALAR specifies. The value of a COLLECTION object is a set of either SCALAR or COLLECTION objects as the specific subclass of the COLLECTION requires.



**Figure 3-44** Data Object Class Hierarchy

The object management classes of DDM are objects that manage the space, existence, and access of data object instances (see Figure 3-45).



**Figure 3-45** Object Management Classes Hierarchy

The management control classes of DDM are managers of managers and therefore, implement data management servers (see Figure 3-46 (on page 441)).

SERVER . . .

**Figure 3-46** Manager Control Hierarchy

**Note:** The architects manually implemented and verified the inheritance of structure and characteristics to enhance this DDM Reference.

**SEE ALSO**

<b>Semantic</b>	<i>CLASS</i> (on page 158) <i>CONCEPTS</i> (on page 243)
<b>sprcls</b>	<i>CLASS</i> (on page 158)

**NAME**

INHERITED — Inherited Definitions Attribute

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'0049'

**Length** \*

**Class** CLASS

**Sprcls** OBJECT - Self-identifying Data

Inherited Definitions Attribute (INHERITED) only applies to class variables that are instances of the class DEFLST (definition list). These variables define a list of named structures or capabilities of classes or instances of classes. Examples of named structures are class variables and instance variables. Examples of named capabilities are class commands and instance commands.

Each named structure or capability is defined through a specified list of attributes. The definition list of a variable with the INHERITED attribute is propagated to subclasses of the class. The definition is copied to the variable with the same name in the subclasses according to the following rules:

- Copy definitions from the superclass to the subclass in the order which the superclass specifies.
- Compare the definition names specified in the subclass to the identically named definitions in the superclass. If a name is not explicitly specified, use the codepoint of the CLASS attribute value as the name.
- If the definitions match, merge them on an attribute-by-attribute basis with the attributes from the subclass definition overriding those from the superclass definition.
- If the definitions do not match, add the subclass definition to the end of the definitions list copied from the superclass.
- Inheritance of definitions proceeds to subclasses of the subclass following the same rules.

As a matter of convenience, DDM specifications show the results of inheritance in each CLASS. However, this would result in excessive repetition in regard to the inheritance of class variables. Therefore, unless additional class variables are being defined, the *lsvr* of a CLASS only specifies the INHERITED attribute. For example, the *lsvr* of this CLASS specifies only the INHERITED attribute because only the variables are inherited from the superclasses of this CLASS.

<b>clsvar</b>	NIL
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
<b>length</b>	*
<b>class</b>	X'0049'
<b>clscmd</b>	NIL
<b>inscmd</b>	NIL



**SEE ALSO**

<b>clscmd</b>	<i>CLASS</i> (on page 158)
<b>clsvar</b>	<i>CLASS</i> (on page 158)
<b>inscmd</b>	<i>CLASS</i> (on page 158)
<b>insvar</b>	<i>CLASS</i> (on page 158)
<b>Semantic</b>	<i>CLASS</i> (on page 158)
<b>sprcls</b>	<i>CLASS</i> (on page 158)

**NAME**

INSTANCE\_OF — Instance of

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'005D'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Instance of (INSTANCE\_OF) String has the codepoint of a class object as its value. It indicates that the value of the variable described is an instance of the specified class.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	6	
class	X'005D'	
value	INSTANCE_OF NOTE REQUIRED	CODPNTDR - Codepoint Data Representation Must be the codepoint of a class object.
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

None.

**NAME**

INTRDBRQS — Interrupt RDB Request

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2003'**Length** \***Class** CLASS**Sprcls** COMMAND - Command <QDDPRMD>

The Interrupt RDB Request (INTRDBRQS) command requests that the target SQLAM interrupt (terminate) a DDM command that is currently being executed for the requester.

The target agent uses information in the RDBINTTKN parameter to route the command to the appropriate target SQLAM instance executing the DDM command being interrupted.

The RDBINTTKN parameter specifies the interrupt token that the requester received on a previous ACCRDBRM.

Requesters cannot request interruption of a DDM command execution on the same DDM conversation previously used to access a relational database (RDB) manager. As a result, the requester might initiate a separate DDM conversation to the target server in order to issue the command.

The target SQLAM validates the specified RDBINTTKN and ensures that the requester is authorized to interrupt the request. A normal command completion reply message (CMDCMPRM) is returned that the target SQLAM's interruption of the DDM command will be successful.

If the target SQLAM determines either that:

- There is no DDM command to interrupt (for example, the DDM command finished execution before the target SQLAM could perform the interrupt request).
- The DDM command to be interrupted is a RDBCMM or RDBRLLBCK command.
- The DDM command cannot be interrupted.

Then the interrupt request is ignored, and the normal command completion reply message (CMDCMPRM) is returned if not already.

The target SQLAM must interrupt the execution of the requester's current DDM command in such a manner that:

- The execution of the DDM command is terminated.
- All effects of the DDM command are removed, and the database returns to the same logical state that existed before the DDM command.
- The database does not rollback the requester's current logical unit of work. Only the effects of the single interrupted DDM command are rolled back, not the entire logical unit of work. After the interrupt process, the requester must be able to commit the current logical unit of work.

The interruption (termination) of the DDM command is reported to the requester through the normal reply mechanisms.

**Exceptions**

If the interrupt token is invalid or the requester is not authorized to interrupt the DDM command execution, then the command is rejected with INTTKNRM.

If the SNA network connection was allocated with synchronization level other than NONE, then the command is rejected with MGRDEPRM.

If the Target agent associated with the DDM conversation has obtained access to a target SQLAM (has successfully processed an ACCRDB command) when the INTRDDBRQS command is received, then the command is rejected with RDBACCRM.

**DSS Carrier: RQSDSS**

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>INSTANCE VARIABLES</b>	
<b>length</b>	*	
<b>class</b>	X'2003'	
<b>rdbinttkn</b>	INSTANCE_OF REQUIRED CMDTRG MINLVL	RDBINTTKN - RDB Interrupt Token <QDDRDBD>   7
<b>rdbnam</b>	INSTANCE_OF OPTIONAL	RDBNAM - Relational Database Name <QDDRDBD>
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>cmddta</b>	<b>NIL</b>	
<b>rpydta</b>	<b>NIL</b>	
<b>cmdrpy</b>	<b>COMMAND REPLIES</b>	
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error <QDDBASD>
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command <QDDBASD>
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check <QDDBASD>
X'124B'	INSTANCE-OF	CMDCMPRM - Command Processing Completed <QDDBASD>
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported <QDDBASD>
X'2210'	INSTANCE_OF	INTTKNRM - Interrupt Token Invalid <QDDRDBD>
X'1218'	INSTANCE_OF	MGRDEPRM - Manager Dependency Error <QDDBASD>
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error <QDDBASD>
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported <QDDBASD>
X'2207'	INSTANCE_OF	RDBACCRM - RDB Currently Accessed <QDDRDBD>
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limits Reached <QDDBASD>
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error <QDDBASD>
X'125F'	INSTANCE-OF	TRGNSPRM - Target Not Supported <QDDBASD>

X'1252'          INSTANCE\_OF    VALNSPRM - Parameter Value Not Supported  
<QDDBASD>

**SEE ALSO**

**insvar**          *ACCRDBRM* (on page 48)  
                  *CMDTRG* (on page 179)  
                  *QDDRDBD* (on page 657)  
                  *RDBINTTKN* (on page 733)  
                  *RDBNAM* (on page 736)

**cmdrpy**          *AGNPRMRM* (on page 65)  
                  *CMDATHRM* (on page 171)  
                  *CMDCHKRM* (on page 173)  
                  *CMDCMPRM* (on page 175)  
                  *CMDNSPRM* (on page 176)  
                  *INTTKNRM* (on page 448)  
                  *MGRDEPRM* (on page 505)  
                  *PRCCNVRM* (on page 625)  
                  *PRMNSPRM* (on page 633)  
                  *QDDBASD* (on page 651)  
                  *QDDRDBD* (on page 657)  
                  *RDBACCRM* (on page 724)  
                  *RSCLMTRM* (on page 778)  
                  *SYNTAXRM* (on page 989)  
                  *TRGNSPRM* (on page 1022)  
                  *VALNSPRM* (on page 1057)

**Semantic**        *ACCRDB* (on page 42)  
                  *ACCRDBRM* (on page 48)  
                  *CMDCMPRM* (on page 175)  
                  *INTTKNRM* (on page 448)  
                  *MGRDEPRM* (on page 505)  
                  *QDDPRMD* (on page 654)  
                  *QDDRDBD* (on page 657)  
                  *RDBACCRM* (on page 724)  
                  *RDBCMM* (on page 728)  
                  *RDBINTTKN* (on page 733)  
                  *RDBRLLBCK* (on page 745)  
                  *RQSDSS* (on page 774)  
                  *SQLAM* (on page 847)

**NAME**

INTTKNRM — Interrupt Token Invalid

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2210'  
**Length** \*  
**Class** CLASS  
**Sprcls** RPYMSG - Reply Message <QDDBASD>

Interrupt Token Invalid (INTTKNRM) Reply Message indicates the target SQLAM has determined that the specified RDBINTTKN value is invalid because of one of the following:

- The token value does not match the interrupt token value returned to the requester on the ACCRDBRM.
- The requester is not authorized to interrupt the execution of a DDM command.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'2210'	
<b>rdbinttkn</b>	INSTANCE_OF REQUIRED MINLVL	RDBINTTKN - RDB Interrupt Token <QDDRDBD> 7
<b>rdbnam</b>	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name <QDDRDBD>
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information <QDDBASD>
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code <QDDBASD> 8 - ERROR - Error Severity Code <QDDPRMD>
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** QDDBASD (on page 651)  
 QDDPRMD (on page 654)  
 QDDRDBD (on page 657)  
 RDBINTTKN (on page 733)  
 SVRCOD (on page 911)  
 SRVDGN (on page 873)

**cmdrpy** INTRDBRQS (on page 445)

**Semantic** ACCRDBRM (on page 48)  
 INTRDBRQS (on page 445)  
 QDDBASD (on page 651)  
 QDDRDBD (on page 657)

*RDBINTTKN* (on page 733)  
*RPYDSS* (on page 766)  
*RPYMSG* (on page 770)  
*SQLAM* (on page 847)

**NAME**

IPADDR — TCP/IP Address

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD  
**Codepoint** X'11E8'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

TCP/IP Address (IPADDR) is the host address and port number for the RDB server. The first four bytes of IPADDR is the binary IP address followed by two bytes binary TCP port number.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	10	
class	X'11E8'	
tcpaddr	INSTANCE_OF LENGTH REQUIRED NOTE	BINDR - Binary Number Field 32  Binary 32 representation of IP address.
tcpport	INSTANCE_OF LENGTH REQUIRED NOTE	BINDR - Binary Number Field 16  Binary 16 representation of TCP port number.
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** ACCRDBRM (on page 48)  
 SRVLSRV (on page 876)  
 SYNCLOG (on page 922)  
**Semantic** SRVLSRV (on page 876)



**NAME**

ISODATFMT — ISO Date Format

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2429'**Length** \***Class** codpnt

ISO Date Format (ISODATFMT) specifies that dates which appear in the SQL statements are in the International Standard Organization (ISO) date format, which is:

yyyy-mm-dd

where - is a hyphen (with the Graphic Character Global Identifier (GCGID) SP10). For more information on GCGID, see *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO****insvar** *STTDATFMT* (on page 894)**Semantic** *STTDATFMT* (on page 894)

**NAME**

ISOLVLALL — Isolation Level All

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2443'**Length** \***Class** codpnt

Isolation Level All (ISOLVLALL) specifies that the execution of SQL statements in the package is isolated (protected) from the actions of concurrent users for rows the requester reads and changes. Specifically, this means:

- Rows that the requester reads, updates, deletes, or inserts are protected from updates by concurrent users until the end of the current unit of work.
- Rows that the requester reads through a read-only database cursor may not be the results of uncommitted update, delete, or insert operations concurrent users perform.
- The results table of an OPNQRY that the requester executes again for the same package section during a single unit of work may be different than the earlier results table. Rows that were inserted or updated by concurrent users or by the requester may show up in the new result table.

**SEE ALSO****insvar** *PKGISOLVL* (on page 592)**Semantic** *PKGISOLVL* (on page 592)

**NAME**

ISOLVLCHG — Isolation Level Change

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2441'**Length** \***Class** codpnt

Isolation Level Change (ISOLVLCHG) specifies that the execution of SQL statements in the package is isolated (protected) from the actions of concurrent users for changes the requester makes. Specifically, this means:

- Rows the requester reads through a read-only database cursor may be the results of uncommitted update, delete, or insert operations concurrent users perform.
- Rows a read-only database cursor is positioning are not protected from updates by concurrent users.
- Rows the requester reads through a non-read-only database cursor may not be the results of uncommitted update, delete, or insert operations concurrent users perform.
- Rows a non-read-only database cursor is positioning are protected from updates by concurrent users until the next row is read or until the end of the current unit of work (UOW) if the current row is updated or deleted.
- Rows being updated or deleted are protected from concurrent users until the end of the current UOW.
- The results table of an OPNQRY that the requester is executing again for the same package section during a single unit of work may be entirely different than the earlier results table. Rows that were inserted, updated, or deleted by concurrent users or by the requester may show up in the new results table.

**SEE ALSO****insvar** *PKGISOLVL* (on page 592)**Semantic** *PKGISOLVL* (on page 592)

**NAME**

ISOLV LCS — Isolation Level Cursor Stability

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2442'**Length** \***Class** codpnt

Isolation Level Cursor Stability (ISOLV LCS) specifies that the execution of Structured Query Language (SQL) statements in the package and the current row to which the database cursor is positioned are isolated (protected) from the actions of concurrent users for changes the requester makes. Specifically, this means:

- Rows the requester reads are protected while the requester has a database cursor positioned at that row. As soon as the requester moves the database cursor to another row, concurrent users can change the row to which the database cursor was previously positioned. Rows a requester is changing are protected from updates by concurrent users until the end of the current unit of work.
- Rows that the requester reads and does not update are never the results of uncommitted update operations concurrent users perform. Updates the requester performs are not available to concurrent users until the current unit of work has ended.
- The results table of an OPNQRY that the requester is executing again for the same package section during a single unit of work may be different than the earlier results table. Newly inserted rows and rows changed by concurrent users (but not updated by the requester) may show up in the new result table. Rows that were deleted by concurrent users (but not updated by the requester) may not be present in the new results table.

**SEE ALSO****insvar** PKGISOLVL (on page 592)**Semantic** PKGISOLVL (on page 592)

**NAME**

ISOLVLNC — Isolation Level No Commit

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2445'**Length** \***Class** codpnt

Isolation Level No Commit (ISOLVLNC) is used when SQL statements are executed.

Commitment control is not used. COMMIT and ROLLBACK statements are not required for changes to be visible to concurrent users.

For SQL statements, this parameter value specifies that the execution of Structured Query Language (SQL) statements in the package is not isolated (protected) from the actions of concurrent users for changes the requester makes. Specifically, this means:

- Rows the requester reads through a read-only database cursor may be the results of uncommitted update, delete, or insert operations concurrent users perform.
- Rows a read-only database cursor is positioning are not protected from updates by concurrent users.
- Rows the requester reads through a non-read-only database cursor may not be the results of uncommitted update, delete, or insert operations concurrent users perform.
- Rows a non-read-only database cursor is positioning are protected from updates by concurrent users until the next row is read or until the current row is updated or deleted.
- Rows being updated or deleted are not protected from concurrent users.
- The results table of an OPNQRY that the requester is executing again for the same package section during a single unit of work may be entirely different from the earlier results table. Rows that concurrent users or requesters insert, update, or delete may show up in the new results table.

**Note:** If ISOLVLNC is specified and the relational database (RDB) does not promote it to a higher isolation level, then updates and deletes are not treated as committable actions. For instance, the RDBUPDRM is not returned.

**SEE ALSO****insvar** *PKGISOLVL* (on page 592)**Semantic** *PKGISOLVL* (on page 592)

**NAME**

ISOLVLRR — Isolation Level Repeatable Read

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2444'**Length** \***Class** codpnt

Isolation Level Repeatable Read (ISOLVLRR) specifies that the execution of Structured Query Language (SQL) statements in the package is isolated (protected) from the actions of concurrent users for rows the requester reads and changes, as well as phantom rows. Specifically this means:

- Rows that the requester reads or changes are protected from updates by concurrent users until the end of the current unit of work.
- Rows that the requester reads but does not update are never the results of uncommitted update operations concurrent users perform. Updates the requester performs are not available to concurrent users until the current unit of work has ended.
- The results table of an OPNQRY that the requester is executing again for the same package section during a single unit of work will always be the same as the earlier results table (except that it may contain updates the requester performs).

**SEE ALSO****insvar** *PKGISOLVL* (on page 592)**Semantic** *PKGISOLVL* (on page 592)

**NAME**

ISOTIMFMT — ISO Time Format

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'242E'**Length** \***Class** CODPNT

The ISO Time Format (ISOTIMFMT) specifies that when times appear in the Structured Query Language (SQL) statements they will be in the International Standard Organization (ISO) time format, which is:

hh.mm.ss

where "." is a period (with the Graphic Character Global Identifier (GCGID) SP11). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO****insvar** *STTTIMFMT* (on page 900)**Semantic** *HMSPRDTIMFMT* (on page 434)  
*STTTIMFMT* (on page 900)

**NAME**

JISDATFMT — Japanese Industrial Standard Date Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'242C'

**Length** \*

**Class** codpnt

Japanese Industrial Standard Date (JISDATFMT) specifies that date values in the Structured Query Language (SQL) statements are in the Japanese Industrial Standard date format (JISDATFMT):

yyyy-mm-dd

where "-" is a hyphen (with the Graphic Character Global Identifier (GCGID) SP10). For more information on GCGID, see *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar** *STTDATFMT* (on page 894)

**Semantic** *STTDATFMT* (on page 894)



**NAME**

JISTIMFMT — Japanese Industrial Standard Time Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2431'

**Length** \*

**Class** CODPNT

Japanese Industrial Standard Time Format (JISTIMFMT) specifies that time values in the Structured Query Language (SQL) statements are in the Japanese Industrial Standard time format (JISTIMFMT):

hh:mm:ss

where ":" is a colon (with the Graphic Character Global Identifier (GCGID) SP13). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar** *STTIMFMT* (on page 900)

**Semantic** *HMSCLNTIMFMT* (on page 432)

**NAME**

JULBLKDATFMT — Julian with Blank Separator Date Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'242D'

**Length** \*

**Class** CODPNT

The Julian with Blank Separator Date Format (JULBLKDATFMT) specifies that dates in the Structured Query Language (SQL) statements are in the following Julian date format:

yy ddd

where the " " separator is a blank (with the Graphic Character Global Identifier (GCGID) SP01). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar** *STTDATFMT* (on page 894)

**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)

**NAME**

JULCMADATFMT — Julian with Comma Separator Date Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'243F'

**Length** \*

**Class** CODPNT

The Julian with Comma Separator Date Format (JULCMADATFMT) specifies that dates in the Structured Query Language (SQL) statements are in the following Julian date format:

*yy, ddd*

where the "," separator is a comma (with the Graphic Character Global Identifier (GCGID) SP08). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar** *STTDATFMT* (on page 894)

**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)

**NAME**

JULHPNDATFMT — Julian with Hyphen Separator Date Format

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2440'**Length** \***Class** CODPNT

The Julian with Hyphen Separator Date Format (JULHPNDATFMT) specifies that dates in the Structured Query Language (SQL) statements are in the following Julian date format:

yy-ddd

where the "-" separator is a hyphen (with the Graphic Character Global Identifier (GCGID) SP10). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO****insvar** *STTDATFMT* (on page 894)**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)

**NAME**

JULPRDDATFMT — Julian with Period Separator Date Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2446'

**Length** \*

**Class** CODPNT

The Julian with Period Separator Date Format (JULPRDDATFMT) specifies that dates in the Structured Query Language (SQL) statements are in the following Julian date format:

yy.ddd

where the "." separator is a period (with the Graphic Character Global Identifier (GCGID) SP11). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar** *STTDATFMT* (on page 894)

**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)

**NAME**

JULSLHDATFMT — Julian with Slash Separator Date Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2447'

**Length** \*

**Class** CODPNT

The Julian with Slash Separator Date Format (JULSLHDATFMT) specifies that dates in the Structured Query Language (SQL) statements are in the following Julian date format:

yy/ddd

where the "/" separator is a period (with the Graphic Character Global Identifier (GCGID) SP12). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar** *STTDATFMT* (on page 894)

**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)

**NAME**

KERSEC - Kerberos Security

**DESCRIPTION (Semantic)****Dictionary** QDDPRMD**Length** \***Class** CONSTANT

The Kerberos Security Mechanism (KERSEC) specifies a security mechanism combination for the ACCSEC command.

See the following terms for the specific meaning and function of the security mechanism:

KERSECOVR Kerberos Security Overview (see *KERSECOVR* (on page 466))

KERSECTKN Kerberos Security Token (see *KERSECTKN* (on page 471))

---

value	11
-------	----

**SEE ALSO**

**rpydta** *ACCSEC* (on page 52)

**insvar** *SECCHK* (on page 800)  
*SECMEC* (on page 811)

**Semantic** *ACCSEC* (on page 52)  
*KERSECOVR* (on page 466)  
*SECCHK* (on page 800)

## NAME

KERSECOVR - Kerberos Security Overview

## DESCRIPTION (Semantic)

Dictionary QDDTTRD

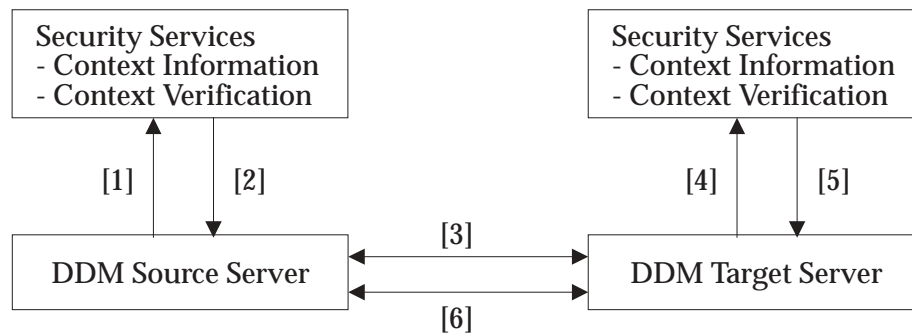
Length \*

Class HELP

The Kerberos Security Overview (KERSECOVR) provides an overview of the DDM flows while the using the Kerberos security mechanism.

Figure 3-47 indicates the DDM commands and replies that flow in the normal process of establishing a connection while using the Kerberos security mechanism for identification and authentication. It shows an example of the exchange of message to obtain and use the security context information within the security token. See the following references for more information on the APIs used by Kerberos:

- IETF RFC 1508, Generic Security Service Application Program Interface (GSS-API)
- IETF RFC 1510, The Kerberos Network Authentication Service (V5)
- IETF RFC 1964, Kerberos Mechanism Type
- Microsoft Security Support Provider Interface (SSPI), which is semantically similar to GSS-API



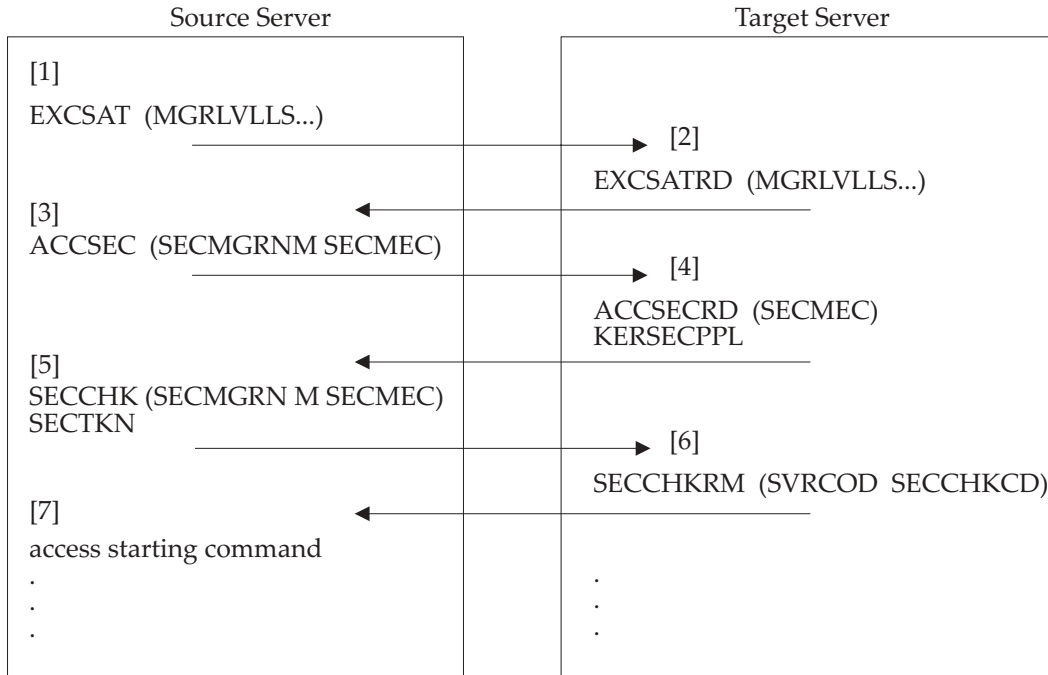
**Figure 3-47** Example of Kerberos-Based Flow using SSPI and GSS-API

The following is a brief description of the example Kerberos flow shown in Figure 3-47. This example shows a source server that utilizes the SSPI-API and a target server that uses the GSS-API.

1. 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 not 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 receive 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.



**Figure 3-48** DDM Kerberos Security Flows

Figure 3-48 describes the synchronization of security information for a source server and target server pair. If a source server is connecting to multiple target servers, the security information shared between each source server and target server pair is independent from each of the other pairs. The following is a brief explanation of the DDM command parameters shown in Figure 3-48. See the appropriate terms for a detailed description of the parameters.

1. The source server identifies the level of security it would like to operate at by identifying the SECMGR at manager level 5 on the EXCSAT command.

```
EXCSAT (
MGRLVLS (
MGRLVL (SECMGR, 5)
.
.
.) )
```

2. The target server receives the EXCSAT command and builds a reply data object with the SECMGR at manager level 5 indicating it can operate at that security level. The target server sends the EXCSATRD reply data to the source server.

```

EXCSATRD (
MGRLVLLS (
MGRLVL (SECMGR, 5)
.
.
.) )

```

3. The source server receives the EXCSATRD reply data. The type of identification and authentication mechanism is negotiated through the ACCSEC command.

The SECMEC parameter indicates the security mechanism to use. This example assumes that the KERSEC security mechanism is specified.

4. The target server receives the ACCSEC command. It supports the security mechanism identified in the SECMEC parameter and returns the value in the ACCSECRD.

If the target server does not support or accept the security mechanism specified in the SECMEC parameter on the ACCSEC command, the target server returns the security mechanism combination values that it does support in the SECMEC parameter in the ACCSECRD object.

Regardless of whether KERSEC has been specified on the ACCSEC command, if KERSEC is returned as a supported security mechanism on the ACCSECRD, the target server may optionally return its Kerberos principal in a KERSECPPL reply data object following the ACCSECRD if it is operating at SECMGR level 7.

5. The source server receives the ACCSECRD object and generates the security token containing the security context information required for security processing.

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

The source server passes the security context information in a SECCHK command with a SECTKN object. For information about the Kerberos security context information, see *KERSECTKN* (on page 471).

6. The target 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 target server may either process the security context information itself or it may call a security resource manager to process the security context information.

The target server generates a SECCHKRM to return to the source server. 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.

7. The source server receives the SECCHKRM.

Assuming security processing is successful, the source server sends a data access starting command to the target 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 security processing fails, the source server might attempt recovery before returning to the application. For example, if the context information has expired, the source server could request new security context information to send to the target server. If the error

condition is not recoverable, the source server returns to the application with an error indicating a security verification failure.

**SEE ALSO**

<b>insvar</b>	<i>SECCHK</i> (on page 800)
<b>Semantic</b>	<i>KERSECPPL</i> (on page 470) <i>SECMGR</i> (on page 814)

## NAME

KERSECPPL — Kerberos Security Principal

## DESCRIPTION (Semantic)

**Dictionary** QDDBASD**Codepoint** X'1C02'**Length** \***Class** CLASS**Sprcls** STRING - String

The Kerberos Security Principal (KERSECPPL) is used for carrying the Kerberos principal. The value is a character string in the CCSID of the CCSIDMGR negotiated at EXCSAT (or CCSID 500 if CCSIDMGR is not supported).

See *KERSECOVR* (on page 466) for information on the DDM flows that carry the KERSECPPL.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'1C02'	
<b>value</b>	INSTANCE_OF	CHRSTRDR - Character String Data Representation
	REQUIRED	
	MINLVL	7
	MINLEN	1
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

## SEE ALSO

**rpydta** ACCSEC (on page 52)**Semantic** ACCSEC (on page 52)  
KERSECOVR (on page 466)

**NAME**

KERSECTKN — Kerberos Security Token

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

The Kerberos Security Token (KERSECTKN) is information provided and used by the Kerberos security mechanism. See *KERSECOVR* (on page 466) for information on the DDM flows that carry the KERSECTKN. The KERSECTKN is carried in a SECTKN object as command and reply data to the SECCHK command.

The security token contains Kerberos security context information to identify and authenticate a user to the target server or to authenticate the target server to the source server.

DDM architecture treats the KERSECTKN as a byte string. The SECMGR is aware of and has knowledge of the contents of the KERSECTKN. The KERSECTKN contains the Kerberos security ticket and security information. DDM does not know or care about the specific contents of this field. If the reader needs to know the actual contents, see the documentation in the overview term.

**SEE ALSO**

None.

NAME

LENGTH — Length of Value Attribute

DESCRIPTION (Semantic)

**Dictionary** QDDPRMD  
**Codepoint** X'001F'  
**Length** \*  
**Class** CLASS  
**Sprcls** BIN - Binary Integer Number

Length (LENGTH) Binary Integer Number specifies that the attribute value is the length of a specified term in the number of units required for representing it.

The unit of measurement varies according to the class of term being described. For example, the unit of a bit string (BITSTRDR) is a bit (BITDR), while the unit of a character string (CHRSTRDR) is a character (CHRDR).

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'001F'	
value	INSTANCE_OF	BINDR - Binary Number Field
	ENULEN	16
	ENULEN	32
	ENULEN	48
	ENULEN	64
	REQUIRED	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

SEE ALSO

**clsvar** CLASS (on page 158)  
**insvar** BIN (on page 118)  
 BOOLEAN (on page 142)  
 BYTDR (on page 145)  
 CCSIDDBC (on page 148)  
 CCSIDMBC (on page 149)  
 CCSIDSBC (on page 154)  
 CMMTYP (on page 183)  
 CNNTKN (on page 220)  
 CODPNTDR (on page 235)  
 DCTINDEN (on page 259)  
 DECPRC (on page 266)  
 DEPERRC (on page 270)  
 DGRIOPRL (on page 279)  
 DSCERRCD (on page 296)  
 DSSFMT (on page 318)  
 FDODSCOFF (on page 407)  
 FDODTAAOFF (on page 409)

*FDOPRMOFF* (on page 413)  
*FDOTRPOFF* (on page 414)  
*FORGET* (on page 423)  
*HEXDR* (on page 428)  
*IPADDR* (on page 450)  
*LOGNAME* (on page 484)  
*LOGTSTMP* (on page 485)  
*MAXBLKEXT* (on page 494)  
*MAXRSLCNT* (on page 497)  
*MAXSCTNBR* (on page 498)  
*MGRVLN* (on page 509)  
*MINLVL* (on page 518)  
*NBRROW* (on page 530)  
*OBJDSS* (on page 536)  
*OBJECT* (on page 540)  
*OUTEXP* (on page 571)  
*OUTOVROPT* (on page 576)  
*PKGATHRUL* (on page 581)  
*PKGCNSTKN* (on page 588)  
*PKGID* (on page 591)  
*PKGNAME* (on page 594)  
*PKGNAMECSN* (on page 596)  
*PKGNAMECT* (on page 598)  
*PKGSN* (on page 606)  
*PRCCNVCD* (on page 621)  
*QRYBLKSZ* (on page 678)  
*QRYROWNBR* (on page 694)  
*RDBALWUPD* (on page 726)  
*RDBCMTOK* (on page 731)  
*RDBCOLID* (on page 732)  
*RDBNAME* (on page 736)  
*RLSCONV* (on page 765)  
*RPYDSS* (on page 766)  
*RQSCRR* (on page 772)  
*RQSDSS* (on page 774)  
*RSLSETFLG* (on page 783)  
*RSNCOD* (on page 786)  
*RTNEXTDTA* (on page 792)  
*RTNSQLDA* (on page 795)  
*SECCHKCD* (on page 806)  
*SECMEC* (on page 811)  
*SECMGRNM* (on page 818)  
*SNAADDR* (on page 827)  
*SQLCSRHLD* (on page 858)  
*SQLSTNBR* (on page 870)  
*SRVLCNT* (on page 875)  
*SRVPRTY* (on page 883)  
*SVCERRNO* (on page 910)  
*SVRCOD* (on page 911)  
*SYNCCTL* (on page 915)  
*SYNERRCD* (on page 985)  
*TYPYSQLDA* (on page 1034)

	<i>UOWDSP</i> (on page 1037)
	<i>UOWID</i> (on page 1038)
<b>mgrlvln</b>	<i>MANAGER</i> (on page 491)
<b>Semantic</b>	<i>APPCMNFL</i> (on page 68)
	<i>APPCMNI</i> (on page 72)
	<i>APPCMNT</i> (on page 76)
	<i>APPSRCCD</i> (on page 79)
	<i>APPTRGER</i> (on page 95)
	<i>ARRAY</i> (on page 101)
	<i>ATTLST</i> (on page 105)
	<i>BINDR</i> (on page 120)
	<i>DEFLST</i> (on page 268)
	<i>MGRLVL</i> (on page 506)
	<i>OBJECT</i> (on page 540)
	<i>OBJOVR</i> (on page 544)
	<i>SYNCMNBK</i> (on page 924)
	<i>SYNCMNCM</i> (on page 926)
	<i>SYNCMNFL</i> (on page 928)
	<i>SYNCMNI</i> (on page 931)
	<i>SYNCMNT</i> (on page 935)



**NAME**

LMTBLKPRC — Limited Block Protocol

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2417'**Length** \***Class** CODPNT

Limited Block Query Protocol (LMTBLKPRC) documents the limited block query protocols. Limited block protocols are used for queries and for the return of result sets generated by the invocation of a stored procedure which are not enabled for rowset positioning. For more information about result sets which use the Limited Block Query Protocol, see *EXCSQLSTT* (on page 381).

For queries with ambiguous non-scrollable cursors and for insensitive scrollable cursors (QRYATTSCR is TRUE and QRYATTSNS is QRYINS), limited block query protocols are used if all of the following are true:

- The OPNQRY command did not specify FRCFIXROW in the QRYBLKCTL parameter.
- The package QRYBLKCTL parameter is LMTBLKPRC.

For queries with non-ambiguous non-scrollable cursors, limited block query protocols are used if all of the following are true:

- The database cursor is declared FETCH ONLY, if it cannot be the target of a WHERE CURRENT OF clause on an SQL UPDATE or DELETE statement.
- The OPNQRY command did not specify FRCFIXROW in the QRYBLKCTL parameter.
- The package QRYBLKCTL parameter is not FRCFIXROW.

For queries with ambiguous sensitive scrollable cursors (QRYATTSCR is TRUE and QRYATTSNS is QRYSNSxxx), limited block query protocols are used if all of the following are true:

- The OPNQRY command did not specify FRCFIXROW in the QRYBLKCTL parameter.
- The package QRYBLKCTL parameter is LMTBLKPRC.
- The cursor does not have dynamic membership (QRYATTSNS is QRYSNSDYN).

For queries with non-ambiguous sensitive scrollable cursors (QRYATTSCR is TRUE and QRYATTSNS is QRYSNSxxx), limited block query protocols are used if all of the following are true:

- The database cursor is declared FETCH ONLY, if it cannot be the target of a WHERE CURRENT OF clause on an SQL UPDATE or DELETE statement.
- The OPNQRY command did not specify FRCFIXROW in the QRYBLKCTL parameter.
- The package QRYBLKCTL parameter is not FRCFIXROW.
- The cursor does not have dynamic membership (QRYATTSNS is QRYSNSDYN).

If the cursor is scrollable (QRYATTSCR is TRUE), the application requester may either use the cursor in a scrollable or non-scrollable manner as follows (see the DRDA Reference, Appendix B for more details):

- For application requesters that wish to provide support for scrollable cursors, a QRYROWSET value must be provided on every CNTQRY command for the scrollable cursor. This indicates that the cursor will be used in a scrollable manner. This allows more than one single-row FETCH to be requested by the target SQLAM and for the rows retrieved to be returned as members of a rowset. If a QRYROWSET value is specified, the application requester is responsible for managing cursor position differences that may result (see the DRDA Reference, Appendix B for more details).
- For application requesters that do not wish to support scrollable cursors, the scrollable cursor can be accessed in a non-scrollable fashion with subsequent CNTQRY commands that do not specify any scrolling parameters.

The OPNQRY command initiates the query process. After each OPNQRY or CNTQRY command, the query is suspended unless some condition terminates the query (see *OPNQRY* (on page 555)). The CNTQRY command can continue a suspended query so that the next portion of the answer set data is returned. A query is terminated any time the CLSQRY command suspends it.

The application server may not select this query protocol if the query returns any LOB data columns that are subject to externalization and the application requester has specified that output overrides may be sent with each CNTQRY command.

The normal response to an OPNQRY command is:

- An OPNQRYRM is returned.
- A description of the query answer set data (see *QRYDSC* (on page 685) and also the DRDA Block Format (BF) rules in the DRDA Reference for detailed information).
- Possibly answer set data is returned as one or more QRYDTA reply data objects following the query description (see the DRDA Query Data Transfer Protocols (QP) rules in the DRDA Reference for detailed information).

If the answer set includes any LOB data whose output form must be resolved at the target when the data is fetched (LOB data *versus* LOB locator form), no QRYDTA is returned until the first CNTQRY command is received.

- Possibly, additional query blocks, limited in number by the value of the MAXBLKEXT parameter of OPNQRY, each containing a QRYDTA reply data object.
- If a QRYROWSET value is specified, the value indicates the number of rows in the explicit DRDA rowset and limits the number of rows to be returned with the CNTQRY command to that number. If a QRYROWSET value is not specified, an implicit value is taken for the DRDA rowset and that number is used to limit the number of rows returned (see the DRDA Query Data Transfer Protocol rule QP4 in the DRDA Reference for detailed information).
- The query or result set is suspended.

The normal response to a CNTQRY command is:

- Query answer set data is returned in one of the following ways:
  - Row data containing the end of at least one row of data is returned. If there is room in the query block, possibly additional row data for other rows, according to the DRDA Block Format (BF) rules (see the DRDA Reference for detailed information).
  - The QRYDTA reply data object is used as a carrier for base row data.
  - The EXTDTA reply data object is used as a carrier for FD:OCA placeholders in the QRYDTA objects. The EXTDTAs are returned in the order of the FD:OCA placeholders

in the QRYDTA.

- EXTDTA data objects for a complete base row sent in response to a previous CNTQRY. This case occurs when the source system requests a RTNEXTDTA value of RTNEXTROW.
- If answer set data is returned, possibly, additional query blocks, limited in number by the value of the MAXBLKEXT parameter of CNTQRY, each containing a QRYDTA reply data object.
 

Any EXTDTA objects associated with the QRYDTA object, in order of the corresponding FD:OCA placeholder in the QRYDTA.
- If a QRYROWSET value is specified, the value indicates the number of rows in the explicit DRDA rowset and limits the number of rows to be returned with the CNTQRY command to that number.
- The query or result set is suspended.

Each QRYDTA reply data object contains one or more answer set rows with associated SQLCAs. If a QRYROWSET value is specified on the CNTQRY command, the value indicates the number of rows in the explicit DRDA rowset and limits the number of rows to be returned with the CNTQRY command to that number. As for the OPNQRY command, the application requester is responsible for managing cursor position differences that may result.

Minimally, each CNTQRY command returns one or more query blocks of QRYDTA objects until the last column of an answer set row is contained in a query block. The one or only query block (which is the first query block that contains the last column of an answer set row) is filled to the specified query block size with additional SQLCAs and answer set rows. This means that several rows, or portions of rows, of answer set data can be placed in a single QRYDTA object or query block.

### Protocol Examples

In the following protocol examples, various possible responses are presented. TYPDEFNAM and TYPDEFOVR reply data objects can precede QRYDSC and/or SQLCARD objects to override the descriptors. To keep the protocols simple, only one example is shown of using TYPDEFNAM and TYPDEFOVR in each of the following protocols.

### Protocols for the OPNQRY Command

In the following protocol lists, everything after the <== symbol represents a set of chained DSSs and (--) represents the parameter list or data content of the DSS. The DSSs may appear on different lines but are to be considered chained to the last DSS on the previous line.

The following examples show some of the valid responses to the OPNQRY command:

- For normal and non-terminating error conditions, an OPNQRYRM is returned followed by a QRYDSC reply data object that contains the description of the answer set data. Answer set data may also be returned in one or more QRYDTA reply data objects.

```
OPNQRY (--) ==>
                <== OPNQRYRM (--) QRYDSC (--)
                --- or ---
OPNQRY (--) ==>
                <== OPNQRYRM (--) SQLCARD (--) QRYDSC (--)
```

```

      --- or ---
OPNQRY (--) ==>
                <== OPNQRYRM (--) SQLCARD (--)
                   QRYDSC (--) QRYDTA (--)

```

- When an OPNQRY command is issued for a query that is currently suspended (opened by a previous OPNQRY command and is not yet terminated), a QRYPOPRM is returned. The query remains suspended.

```

OPNQRY (--) ==>
                <== QRYPOPRM (--)

```

- If the query is non-scrollable and it is at end of data (SQLSTATE 02000), then the target SQLAM may close the query implicitly by returning an ENDQRYRM and an SQLCARD to the source server, based on the cursor type and the QRYCLSIMP value as specified previously on the OPNQRY command.
- Otherwise, if the target SQLAM encounters a query terminating error condition for the cursor, then an ENDQRYRM and an SQLCARD object are included in the reply chain.

```

OPNQRY (--) ==>
                <== OPNQRYRM (--) QRYDSC (--) QRYDTA (--)
                   ENDQRYRM (--) SQLCARD (--)

```

--- or ---

```

OPNQRY (--) ==>
                <== OPNQRYRM (--) SQLCARD (--) QRYDSC (--)
                   QRYDTA (--) ENDQRYRM (--) SQLCARD (--)

```

--- or ---

```

OPNQRY (--) ==>
                <== OPNQRYRM (--) TYPDEFNAM (--) TYPDEFOVR (--)
                   QRYDSC (--) QRYDTA (--)
                   ENDQRYRM (--) TYPDEFNAM (--) TYPDEFOVR (--)
                   SQLCARD (--)

```

- If the target relational database (RDB) detects an error condition preventing the database cursor from being opened prior to the OPNQRYRM being returned, then an OPNQFLRM is returned followed by an SQLCARD object. This terminates the query.

```

OPNQRY (--) ==>
                <== OPNQFLRM (--) SQLCARD (--)

```

- If the target RDB detects an error condition RDB preventing the database cursor from remaining open after the OPNQRYRM is returned, then an ENDQRYRM is returned followed by an SQLCARD object. This terminates the query.

```

OPNQRY (--) ==>
                <== OPNQRYRM (--) QRYDSC (--)
                   <== ENDQRYRM (--) SQLCARD (--)

```

--- or ---

```

OPNQRY (--) ==>
                <== OPNQRYRM (--) QRYDSC (--)
                   <== QRYDTA (--) ENDQRYRM (--) SQLCARD (--)

```

- For terminating error conditions, a reply message is returned. This terminates the query. The reply message always precedes an SQLCARD object.

```
OPNQRY (--) ==>
                <== ABNUOWRM (--) SQLCARD (--)
```

### Protocols for the CNTQRY Command

In the following protocol lists, everything after the <== symbol represents a set of chained DSSs and (--) represents the parameter list or data content of the DSS. The DSSs may appear on different lines but are to be considered chained to the last DSS on the previous line.

The following examples show some of the valid responses to the CNTQRY command:

- For normal and non-terminating error conditions, one or more query blocks containing QRYDTA reply data objects or EXTDTA objects are returned.

If any rows in the answer contain data that will be flowed as externalized FD:OCA data in a DDM EXTDTA object, then the EXTDTA objects associated with a row flow after the QRYDTA object that contains the complete row.

```
CNTQRY (--) ==>
                <== QRYDTA (--)
                --- or ---
CNTQRY (--) ==>
                <== QRYDTA (--) QRYDTA (--) QRYDTA (--)
                --- or ---
CNTQRY (--) ==>
                <== QRYDTA (--) EXTDTA (--) EXTDTA (--) EXTDTA (--)
                --- or ---
CNTQRY (--) ==>
                <== QRYDTA (--) EXTDTA (--) EXTDTA (--) EXTDTA (--)
                <== QRYDTA (--) EXTDTA (--) EXTDTA (--) EXTDTA (--)
                <== QRYDTA (--) EXTDTA (--) EXTDTA (--) EXTDTA (--)
                --- or ---
CNTQRY (--) ==>
                <== EXTDTA (--) EXTDTA (--) EXTDTA (--) EXTDTA (--)
```

- For non-scrollable cursors, if the target SQLAM is aware that the last of the answer set has previously been returned and the query is at end of data (SQLSTATE 02000), subject to the cursor type and the QRYCLSIMP value as specified previously on the OPNQRY command, an ENDQRYRM and an SQLCARD object may be included in the reply chain to indicate the query is closed implicitly by the target SQLAM.
- Otherwise, if the target SQLAM encounters a query terminating error condition for the cursor, then an ENDQRYRM and an SQLCARD object are included in the reply chain.

```
CNTQRY (--) ==>
                <== QRYDTA (--) ENDQRYRM (--) SQLCARD (--)
                --- or ---
```

```
CNTQRY (--) ==>
                <== QRYDTA (--) ENDQRYRM (--) TYPDEFNAM (--)
                TYPDEFOVR (--) SQLCARD (--)
```

```
--- or ---
```

```
CNTQRY (--) ==>
                <== QRYDTA (--) QRYDTA (--) QRYDTA (--)
                ENDQRYRM (--) SQLCARD (--)
```

```
--- or ---
```

```
CNTQRY (--) ==>
                <== ENDQRYRM (--) SQLCARD (--)
```

- If EXTDTA objects are returned with the answer set, the answer set data is not complete until the EXTDTA objects are returned. The ENDQRYRM and SQLCARD may or may not be returned with the EXTDTA, depending on the DRDA Query Termination (QT) rules (see the DRDA Reference for detailed information). If they are returned, they flow as follows:

```
CNTQRY (--) ==>
                <== QRYDTA (--) EXTDTA (--) EXTDTA (--) EXTDTA (--)
                ENDQRYRM (--) SQLCARD (--)
```

```
--- or ---
```

```
CNTQRY (--) ==>
                <== QRYDTA (--) EXTDTA (--) EXTDTA (--) EXTDTA (--)
                <== QRYDTA (--) EXTDTA (--) EXTDTA (--) EXTDTA (--)
                ENDQRYRM (--) SQLCARD (--)
```

```
--- or ---
```

```
CNTQRY (--) ==>
                <== EXTDTA (--) EXTDTA (--) EXTDTA (--)
                ENDQRYRM (--) SQLCARD (--)
```

- For terminating error conditions, a reply message is returned. This terminates the query. The reply message always precedes an SQLCARD object.

```
CNTQRY (--) ==>
                <== ABNUOWRM (--) SQLCARD (--)
```

### Protocols for the CLSQRY Command

The following examples show some of the valid responses to the CLSQRY command:

- If the query is suspended under normal conditions, then an SQLCARD object is returned.

```
CLSQRY (--) ==>
                <== SQLCARD (--)
```

```
--- or ---
```

```
CLSQRY (--) ==>
                <== TYPDEFNAM (--) TYPDEFOVR (--) SQLCARD (--)
```

- If the query is terminated or not open, then the QRYNOPRM is returned.

```
CLSQRY (--) ==>
                <== QRYNOPRM (--)
```

- If the `OPNQRY` or `CNTQRY` command saves a reply message (other than `ENDQRYRM`) and an `SQLCARD` object, then the saved reply message and `SQLCARD` object are returned. This terminates the query.

```
CLSQRYP (--) ==>
                <== ABNUOWRM (--) SQLCARD (--)
```

- If the `OPNQRY` or `CNTQRY` command saves an `ENDQRYRM` and an `SQLCARD` object, then only the saved `SQLCARD` object is returned. The `ENDQRYRM` is discarded.

```
CLSQRYP (--) ==>
                <== SQLCARD (--)
```

- For terminating error conditions that prevent the `CLSQRYP` command from being passed to the target `SQLAM`, an appropriate reply message (RDB not accessed reply message, in the example below) is returned.

```
CLSQRYP (--) ==>
                <== RDBNACRM (--)
```

### SEE ALSO

<b>clsvar</b>	<i>QRYPRCTYP</i> (on page 692)
<b>insvar</b>	<i>CNTQRY</i> (on page 222) <i>QRYBLKCTL</i> (on page 672)
<b>rpydta</b>	<i>OPNQRY</i> (on page 555)
<b>Semantic</b>	<i>CNTQRY</i> (on page 222) <i>ENDQRYRM</i> (on page 342) <i>EXCSQLSTT</i> (on page 381) <i>EXTDTA</i> (on page 397) <i>EXTDTAOVR</i> (on page 399) <i>OPNQRY</i> (on page 555) <i>QRYBLK</i> (on page 670) <i>QRYBLKCTL</i> (on page 672) <i>QRYCLSIMP</i> (on page 680) <i>QRYROWSET</i> (on page 697) <i>RTNEXTDTA</i> (on page 792) <i>SQLAM</i> (on page 847)

## NAME

LOCDATFMT — Local Date Format

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD

**Codepoint** X'2448'

**Length** \*

**Class** CODPNT

The Local Date Format (LOCDATFMT) specifies that dates in the Structured Query Language (SQL) statements are in a locally-defined format. The local date format is determined based on the locale used by the application at the source server. It typically matches one of the other predefined date formats, but it may not.

The means for conveying the relevant locale information of the application from the source server to the target server is implementation-specific and is not architected by DDM. For instance, the source server may achieve this objective by flowing a non-local SQL SET LOCALE statement to the target server.

## SEE ALSO

**insvar** *STTDATFMT* (on page 894)

**Semantic** *CODPNT* (on page 233)  
*DFTDATFMT* (on page 272)  
*QDDRDBD* (on page 657)



**NAME**

LOCTIMFMT — Local Time Format

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2449'**Length** \***Class** CODPNT

The Local Time Format (LOCTIMFMT) specifies that times in the Structured Query Language (SQL) statements are in a locally-defined format. The local time format is determined based on the locale used by the application at the source server. It typically matches one of the other predefined time formats, but it may not.

The means for conveying the relevant locale information of the application from the source server to the target server is implementation-specific and is not architected by DDM. For instance, the source server may achieve this objective by flowing a non-local SQL SET LOCALE statement to the target server.

**SEE ALSO****insvar** *STTTIMFMT* (on page 900)**Semantic** *CODPNT* (on page 233)  
*DFTTIMFMT* (on page 275)  
*QDDRDBD* (on page 657)

## NAME

LOGNAME — Log Name

## DESCRIPTION (Semantic)

**Dictionary** QDDBASD**Codepoint** X'1184'**Length** \***Class** CLASS**Sprcls** STRING - String

Log Name (LOGNAME) specifies the name of the log used by a SYNCPTMGR.

Log names should be unique. It is recommended that the log name be generated by concatenating a unique network name with the resynchronization network address.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	22	
class	X'1184'	
value	INSTANCE_OF	BYTSTRDR - Byte String
	LENGTH	18
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

## SEE ALSO

**insvar** *SYNCLOG* (on page 922)**Semantic** *LOGTSTMP* (on page 485)

**NAME**

LOGTSTMP — Log Timestamp

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1185'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Log Timestamp (LOGTSTMP) specifies a timestamp of when the log specified in LOGNAME was created. The timestamp is an 18-byte character string containing year, month, day, hour, minute, second, and fractions of a second up to tenthousandths of a second in the format YYYYMMDDHHMMSSTTTT.

This identifies the log to be used during sync point operations. The log timestamp may change when the SYNCPTMGR has lost its log and cold started or when the SYNCPTMGR has cold started with another SYNCPTMGR. Refer to *SYNCPTOV* (on page 944).

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	22	
class	X'1185'	
value	INSTANCE_OF LENGTH	CHRSTRDR - Character String 18
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *SYNCLOG* (on page 922)

## NAME

LVLCMP — Level Compatibility

## DESCRIPTION (Semantic)

**Dictionary** QDDTTRD**Length** \***Class** HELP

Level Compatibility (LVLCMP) describes how the higher levels of the DDM architecture are compatible with the lower levels of the architecture. This means that all the functions available in level N are also available in level N+M of the architecture.

A level N+M manager of a source server is compatible with a corresponding level N manager of a target server if the source server does not send any commands, command parameters, command parameter values, or command data objects not defined for level N.

A level N+M manager of a target server is compatible with a corresponding level N manager of a source server if the target server does not send any replies, reply parameters, reply parameter values, or reply data objects not defined for level N.

Level compatibility is an important programming consideration for products that implement the DDM architecture. If the DDM product is an upgraded release of an existing DDM product or if it must be used in existing networks with other DDM products, then the DDM product must evaluate what levels of DDM it must implement. A source server implementing DDM Level 2 may also need to implement DDM Level 1 to ensure data connectivity to existing target servers that have implemented only DDM Level 1. A target server upgrading to DDM Level 2 may need to function with source servers that have implemented only Level 1.

The evaluation of which levels of the DDM architecture a new software product implements is a business decision based on:

- What existing products must have data connectivity with the new software product
- Whether all systems in the existing network should be upgraded when the new software product is introduced or if a transition/migration period is needed
- The cost of implementing more than one level of the DDM architecture
- The number of levels of the DDM architecture to be supported

This is not an exhaustive list, but it points out some of the areas to consider when making level compatibility decisions for a DDM product.

The architecture specifies Architecture Rules for Compatibility and Product Conformance Rules for Compatibility. The Architecture Rules for Compatibility control the changes which can be made to produce a new level of the DDM architecture. The Product Conformance Rules for Compatibility place requirements on existing DDM products being upgraded to a new level of DDM architecture.

### Architecture Rules for Compatibility

The following rules must be adhered to for the architecture to be compatible with lower levels when level N+M of the architecture is being developed:

1. The EXCSAT command cannot have any parameters added to it or removed from it. This allows the initial exchange between a level N+M source server and a level N target server to discover that their managers are at incompatible levels in an orderly (error-free) manner.

New managers can be added to the MGRLVLLS of EXCSAT.

2. A DDM target server must be capable of supporting all defined levels of all managers.
3. A DDM source server must be capable of supporting all defined levels of all managers.
4. New managers can be added to the architecture.

Adding new managers allows the DDM server model to grow in an organized manner to handle major new functional requirements.

5. For Commands:

- New commands can be added to an existing manager. The new commands can be either REQUIRED or OPTIONAL.

This rule allows a manager to grow and meet new, additional requirements.

- Existing commands cannot be removed from existing managers.

This rule ensures that a manager keeps all of its functions in a higher level of the architecture which also existed in a lower level of the architecture.

- Existing commands can be changed from OPTIONAL to REQUIRED for the managers supporting them but cannot be changed from REQUIRED to OPTIONAL.

This rule ensures that all implementations of a manager support at least the same REQUIRED commands as were supported in a lower level of the architecture. In other words, the REQUIRED support a manager provides can be expanded but not decreased in higher levels.

- Existing commands cannot return new reply messages for conditions that existed in a lower level of the architecture. New reply messages can be returned only for new conditions.<sup>10</sup>

This rule prevents new reply messages from being returned when a new function has not been added to a command. This rule intends to make the interface mapping easier for source DDM servers when moving from lower to higher manager levels.

- Existing commands cannot return new reply data objects for conditions or situations that existed in a lower level of the architecture.<sup>11</sup> New reply data objects can be returned for new conditions or situations.

---

10. This rule was violated in DDM Level 2. New reply messages were added to several file-oriented commands to report directory-related errors. The directory error conditions existed in DDM Level 1 but were generally reported with file-oriented reply messages.

11. This rule was violated in DDM Level 2. A RECAL object, with RECCNT, was added to several commands so that identical (repeated) records would only have to be sent once instead of multiple times.

This rule prevents new reply data objects from being returned when a new function has not been added to a command. This rule intends to make the interface mapping easier for source DDM servers when moving from lower to higher manager levels.

- Existing commands cannot remove any command data objects.

This rule helps to ensure that a command keeps all of its functions in a higher level of the architecture that existed in a lower level of the architecture.

#### 6. For Command Parameters:

- New parameters can be added to existing commands. The new parameters can be either REQUIRED or OPTIONAL.

The addition of a REQUIRED parameter does not cause a problem because the agent can select the appropriate parsing tables to use for each command and manager based on the manager level list the EXCSAT command exchanges.

Adding a new REQUIRED parameter to an existing command will add a new parameter that is mutually-exclusive with an existing REQUIRED parameter. In this particular case, the existing parameter would also have the MTLEXC attribute added to it.

Adding a new OPTIONAL parameter to an existing command will add a new optional function. This approach allows new functions similar to those of an existing command to be introduced into the architecture without having to add a whole new command.

- Existing parameters on existing commands cannot be removed.

This rule ensures that a command keeps all of its functions in a higher level of the architecture that existed in a lower level of the architecture.

- Existing parameters can be changed from OPTIONAL to REQUIRED. Existing parameters cannot be changed from REQUIRED to OPTIONAL.

Changing an existing parameter from OPTIONAL to REQUIRED would prevent accidental damage from occurring because the documented default value is determined not to be the most common value used. This would probably only happen for parameters that could cause data to be deleted, overwritten, or changed in some manner.

#### 7. For Parameter Values:

- New enumerated parameter values (ENUVAL) can be added to existing parameters.

New enumerated parameter values may need to be added to existing lists or parameters so that new functions can be added non-disruptively. For example, new managers will be added as enumerated values to the MGRLVL parameter so that the EXCSAT command can exchange manager-level information.

- Existing enumerated parameter values (ENUVAL) of existing parameters cannot be removed.

This rule ensures that a command keeps all of its functions in a higher level of the architecture that existed in a lower level of the architecture.

- Existing default parameter values (DFTVAL) of existing parameters cannot be changed. A default value can be removed if the existing parameter is changed from OPTIONAL to REQUIRED.

This rule provides consistent defaults from one level of the architecture to the next.

- Existing special values (SPCVAl) of existing parameters cannot be changed. New special values can be added to existing parameters.

This rule ensures that a command keeps all of its functions in a higher level of the architecture that existed in a lower level of the architecture.

8. A proposed architecture change cannot violate these Architecture Rules for Compatibility unless unanimous agreement is received from all active DDM implementations at the time of the proposed change.

### **Product Conformance Rules for Compatibility**

1. A DDM product designer may choose any architecture level of a manager for an initial product offering. It does not have to support all defined architecture levels for the manager.
2. A DDM target server must support level N of a manager if it supports level P of the manager and  $S \leq N \leq P$  where level S is the initial level of the manager that the product supported.
3. A DDM source server must support level N of a manager if it supports level P of the manager and  $S \leq N \leq P$  where level S is the initial level of the manager that the product supported.
4. An upgraded, existing DDM product must maintain compatibility for ongoing functions with its previous release levels.
5. An upgraded, existing DDM product can drop support for a function (optional manager, command, parameter, or value) it supported in its previous release.

This rule allows a product to remove functions that the product's customer set does not use. Product designers are not required to support optional functions that are unnecessary for their product. For example, if users of a product are not using the SETPRV command, then the product designer can drop support of the SETPRV command from the RELRNBAM, RNDRNBAM, CMBRNBAM, and CMBACCAM access methods in the next release.

### **Compatibility Documentation**

To make the architecture compatible from level to level, the architecture specifies which functions are available and the semantics required for those functions in each level.

Each term has a status field that indicates the level of the DDM architecture in which the term was introduced and the levels of the architecture in which it was changed.

The MINLVL (minimum-level) attribute has been defined and applied throughout the architecture. The MINLVL attribute applies to the entity that immediately precedes it or (when indented) of which it is an attribute. The manager-level number tests against the MINLVL attribute value to check if the entity with the MINLVL attribute is valid for the selected manager level. If the MINLVL value is less than or equal to the selected manager level, then the entity is valid for the manager. Based on the entity:

Manager name

The named manager is not supported in DDM levels prior to that MINLVL specifies.

Command name

The named command is not supported in DDM levels prior to that MINLVL specifies.

**Command data**

The named command data is not supported in DDM levels prior to that MINLVL specifies.

**Reply message name**

The named reply message is not supported in DDM levels prior to that MINLVL specifies.

**Reply data**

The named reply data is not supported in DDM levels prior to that MINLVL specifies.

**INSTANCE\_OF attribute**

The named parameter is not supported in DDM levels prior to that MINLVL specifies.

**ENUVAL attribute**

The parameter value is not supported in DDM levels prior to that MINLVL specifies.

**Other attributes**

The attribute is not valid in DDM levels prior to that MINLVL specifies.

**SEE ALSO**

**Semantic**      *CONCEPTS* (on page 243)



**NAME**

MANAGER — Resource Manager

**DESCRIPTION (Semantic)****Dictionary** QDDBASD**Codepoint** X'1456'**Length** \***Class** CLASS**Sprcls** OBJECT - Self-identifying Data

Resource Manager (MANAGER) is an object that manages a collection of primitive objects. Examples of resource managers are files, directories, dictionaries, and access methods. Examples of primitive objects are character strings, numbers, names, records, classes, commands, and reply messages.

Primitive objects can exist only within the domain of a resource manager or in a DSS for transmission. Resource managers can only exist within a server.

Primitive objects can combine and interact within a resource manager to form structures of arbitrary complexity. For example, a variety of objects are required to define a CLASS within a DICTIONARY.

A resource manager can impose order or structure over the primitive objects in its domain.

Resource managers can combine and interact to form structures of arbitrary complexity. For example, instances of the following subclasses of MANAGER implement a server:

**AGENT** Represents a requester to a server.

**DICTIONARY** Describes the various classes of managers and primitives available to the server.

**RDB** Controls the data and associated resources of a relational database. This manager was not part of a DDM server prior to Level 3.

**SECMGR** Validates users and authorizes the use and access of resources. The SECMGR can be at DDM Level 1 or DDM Level 5.

**SQLAM** Controls access to a relational database. This manager was not part of a DDM server prior to Level 3.

**SUPERVISOR** Coordinates the resources and activities of the server.

**SYNCPTMGR** Manages two-phase commitment control for recoverable resources. This manager was not part of a DDM server prior to Level 4.

**SYSCMDMGR** Provides the services of processing system commands, including the sending of certain commands to remote systems and the processing of their reply messages. This manager was not part of a DDM server prior to Level 4.

<b>clsvar</b>		<b>CLASS VARIABLES</b>
<b>mgrlvl</b>	INSTANCE_OF TITLE LENGTH NOTE	BIN - Binary Integer Number manager-level number 16 This is the support level of a manager by a server. See MGRVLN for the rules to be followed in setting manager levels.
<b>mgrdepls</b>	INSTANCE_OF TITLE NOTE	DEFLST - Definition List manager dependency list This list documents the dependencies of manager classes on each other. Each entry must specify the CLASS and the MGRVLN of a manager on which the manager being defined has a semantic or functional dependency. A NOTE describing the dependency should be provided. See <i>SUBSETS</i> (on page 902) for a description of the effects of manager-level dependencies on product subset selection.
<b>vldattls</b>	INSTANCE_OF TITLE NOTE	DEFLST - Definition List valid attributes list The list of attributes that can be specified for a manager. The <b>vldattls</b> variable of the manager's class determines the attributes of the class.
<b>insvar</b>		<b>CLASS INSTANCE VARIABLES</b>
length	*	
class	X'1456'	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	
<b>mgrlvl</b>	2	
<b>mgrdepls</b>	NIL	
<b>vldattls</b>		<b>VALID ATTRIBUTES</b>
X'0019'	INSTANCE_OF	HELP - Help Text
X'1452'	INSTANCE_OF NOTE	MGRNAM - Manager Name This is the name of the manager object.
X'0045'	INSTANCE_OF	TITLE - Title

**SEE ALSO**

**Semantic**

- AGENT* (on page 61)
- CCSIDMGR* (on page 150)
- CMNMGR* (on page 196)
- DICTIONARY* (on page 286)
- EXCSAT* (on page 363)
- EXCSATRD* (on page 369)
- INHERITANCE* (on page 437)
- LMTBLKPRC* (on page 475)
- OBJECT* (on page 540)
- RDB* (on page 718)
- RSYNCMGR* (on page 787)
- SECMGR* (on page 814)

*SERVER* (on page 824)  
*SQLAM* (on page 847)  
*STRLYR* (on page 890)  
*SUPERVISOR* (on page 908)  
*SYNCPTMGR* (on page 939)  
*XAMGR* (on page 1063)

NAME

MAXBLKEXT — Maximum Number of Extra Blocks

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'2141'  
**Length** \*  
**Class** CLASS  
**Sprcls** BIN - Binary Integer Number

Maximum Number of Extra Blocks (MAXBLKEXT) specifies a limit on the number of extra blocks of answer set data per result set that the requester is capable of receiving as reply data in the response to an OPNQRY, CNTQRY, or an EXCSQLSTT command that invokes a stored procedure. The number of extra blocks actually returned is dependent on the capabilities of the target SQLAM and the dynamic state of the target SQLAM at the time it executes the command. This parameter is only meaningful when the target SQLAM selects limited block protocol.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	6	
class	X'2141'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	16
	NOTE	A value of N allows the target SQLAM to return up to N extra query blocks of answer set data per result set.
	MINVAL	0
	NOTE	A value of zero indicates that the requester is not capable of receiving extra query blocks of answer set data.
	SPCVAL	-1
	NOTE	A value of minus one indicates that the requester is capable of receiving the entire result set.
	DFTVAL	0
	OPTIONAL	

SEE ALSO

**insvar** CNTQRY (on page 222)  
 EXCSQLSTT (on page 381)  
 OPNQRY (on page 555)  
**Semantic** EXCSQLSTT (on page 381)  
 LMTBLKPRC (on page 475)  
 QRYBLK (on page 670)  
 RSLSETFLG (on page 783)

**NAME**

MAXLEN — Maximum Length Attribute

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD  
**Codepoint** X'0021'  
**Length** \*  
**Class** CLASS  
**Sprcls** BIN - Binary Integer Number

Maximum Length Attribute (MAXLEN) specifies that the attribute value is the maximum length of a term.

The unit of measurement varies according to the type of term being described. For example, the unit of a bit string (BITSTRDR) is a bit (BITDR) while the unit of a character string (CHRSTRDR) is a character (CHRDR).

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'0021'	
<b>value</b>	INSTANCE_OF	BINDR - Binary Number Field
	ENULEN	16
	ENULEN	32
	ENULEN	48
	ENULEN	64
	REQUIRED	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** CNTQRY (on page 222)  
 BNDOPTRM (on page 134)  
 BNDOPTRVL (on page 135)  
 CRRTKN (on page 246)  
 DFTRDBCOL (on page 274)  
 EXCSQLSTT (on page 381)  
 MGRNAM (on page 513)  
 NAMDR (on page 526)  
 NAME (on page 527)  
 NAMSVMDR (on page 528)  
 NEWPASSWORD (on page 531)  
 PASSWORD (on page 578)  
 PKGOWNID (on page 600)  
 PKGRPLVRS (on page 605)  
 PRDDTA (on page 630)  
 PRDID (on page 631)  
 RPYDSS (on page 766)  
 RQSDSS (on page 774)  
 SNAADDR (on page 827)

	<i>SQLJOBNAM</i> (on page 866)
	<i>SRVCLSNM</i> (on page 872)
	<i>SRVDGN</i> (on page 873)
	<i>SRVRLSLV</i> (on page 884)
	<i>TCPHOST</i> (on page 999)
	<i>TEXT</i> (on page 1018)
	<i>TITLE</i> (on page 1020)
	<i>USRID</i> (on page 1044)
	<i>VRSNAM</i> (on page 1059)
<b>semantic</b>	<i>CODPNT</i> (on page 233)
	<i>CONSTANT</i> (on page 244)
	<i>HELP</i> (on page 425)

**NAME**

MAXRSLCNT — Maximum Result Set Count

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2140'  
**Length** \*  
**Class** CLASS  
**Sprcls** BIN - Binary Integer Number

Maximum Result Set Count (MAXRSLCNT) specifies a limit on the number of result sets that the requester is capable of receiving as reply data in the response to an EXCSQLSTT command that invokes a stored procedure. If the stored procedure generates more than MAXRSLCNT result sets, then the target system returns, at most, the first MAXRSLCNT of these result sets. The stored procedure defines the order in which the target system returns result sets.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	6	
<b>class</b>	X'2140'	
<b>value</b>	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	16
	NOTE	A value of N allows the target SQLAM to return up to N result sets.
	MINVAL	0
	NOTE	A value of zero indicates that the requester is not capable of receiving result sets as reply data in the response to EXCSQLSTT.
	SPCVAL	-1
	NOTE	A value of minus one indicates that the requester is capable of receiving all result sets in the response to EXCSQLSTT.
	DFTVAL	0
	OPTIONAL	

**SEE ALSO**

**insvar** EXCSQLSTT (on page 381)  
**Semantic** EXCSQLSTT (on page 381)  
 LMTBLKPRC (on page 475)

NAME

MAXSCTNBR — Maximum Section Number

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD

**Codepoint** X'2127'

**Length** \*

**Class** CLASS

**Sprcls** BIN - Binary Integer Number

Maximum Section Number (MAXSCTNBR) Binary Integer Number specifies the highest section number the source server program preparation process (a pre-compiler) assigns or reserves for this package.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	6	
<b>class</b>	X'2127'	
<b>value</b>	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	16
	MINVAL	1
	DFTVAL	''
	NOTE	The default value is the greater of one, or the highest section number a BNDSQLSTT statement references for the current bind process.
	REQUIRED	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

SEE ALSO

**insvar** ENDBND (on page 336)

**Semantic** ENDBND (on page 336)



**NAME**

MAXVAL — Maximum Value Attribute

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'0022'

**Length** \*

**Class** CLASS

**Sprcls** SCALAR - Scalar Object

Maximum Value Attribute (MAXVAL) Scalar Object specifies that the attribute value is the maximum valid value for a term.

The attribute value specified must have attributes compatible with those specified for the term.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'0022'	
<b>value</b>	SPRCLS NOTE	FIELD - A Discrete Unit of Data The attributes for this variable vary according to what is being defined. Also, when BITDR, BITSTRDR, HEXDR, or HEXSTRDR values are specified, they are represented as being left-justified in the fewest number of whole bytes possible. For example, 123 would be X'1230'.
	REQUIRED	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *OBIDSS* (on page 536)  
*QRYBLKSZ* (on page 678)  
*RPYDSS* (on page 766)  
*RQSDSS* (on page 774)

**NAME**

MDYBLKDATFMT — MDY with Blank Separator Date Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'244A'

**Length** \*

**Class** CODPNT

The MDY (Month, Day, Year) with Blank Separator Date Format (MDYBLKDATFMT) specifies that dates in the Structured Query Language (SQL) statements are in the following MDY date format:

mm dd yy

where the " " separator is a blank (with the Graphic Character Global Identifier (GCGID) SP01). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar** *STTDATFMT* (on page 894)

**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)

**NAME**

MDYCMADATFMT — MDY with Comma Separator Date Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'244B'

**Length** \*

**Class** CODPNT

The MDY (Month, Day, Year) with Comma Separator Date Format (MDYCMADATFMT) specifies that dates in the Structured Query Language (SQL) statements are in the following MDY date format:

mm, dd, yy

where the "," separator is a comma (with the Graphic Character Global Identifier (GCGID) SP08). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar** *STTDATFMT* (on page 894)

**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)

**NAME**

MDYHPNDATFMT — MDY with Hyphen Separator Date Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2451'

**Length** \*

**Class** CODPNT

The MDY (Month, Day, Year) with Hyphen Separator Date Format (MDYHPNDATFMT) specifies that dates in the Structured Query Language (SQL) statements are in the following MDY date format:

mm-dd-yy

where the "-" separator is a hyphen (with the Graphic Character Global Identifier (GCGID) SP10). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar** *STTDATFMT* (on page 894)

**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)

**NAME**

MDYPRDDATFMT — MDY with Period Separator Date Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2452'

**Length** \*

**Class** CODPNT

The MDY (Month, Day, Year) with Period Separator Date Format (MDYPRDDATFMT) specifies that dates in the Structured Query Language (SQL) statements are in the following MDY date format:

mm.dd.yy

where the "." separator is a period (with the Graphic Character Global Identifier (GCGID) SP11). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar** *STTDATFMT* (on page 894)

**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)

**NAME**

MDYSLHDATFMT — MDY with Slash Separator Date Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2453'

**Length** \*

**Class** CODPNT

The MDY (Month, Day, Year) with Slash Separator Date Format (MDYSLHDATFMT) specifies that dates in the Structured Query Language (SQL) statements are in the following MDY date format:

mm/dd/yy

where the "/" separator is a slash (with the Graphic Character Global Identifier (GCGID) SP12). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar** *STTDATFMT* (on page 894)

**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)

**NAME**

MGRDEPRM — Manager Dependency Error

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1218'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

Manager Dependency Error (MGRDEPRM) Reply Message indicates that a request has been made to use a manager, but the requested manager requires specific support from some other manager that is not present. The *deperrcd* parameter shows which manager dependency was not met.

**DSS Carrier:** RPYDSS

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'1218'	
<b>deperrcd</b>	INSTANCE_OF REQUIRED	DEPERRCD - Manager Dependency Error Code
<b>rdbnam</b>	INSTANCE_OF OPTIONAL	RDBNAM - Relational Database Name
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED	SVRCOD - Severity Code
	ENUVAL	8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**cmdrpy** ACCRDB (on page 42)  
 EXCSAT (on page 363)  
 INTRDBRQS (on page 445)  
 SECCHK (on page 800)

**Semantic** ACCRDB (on page 42)  
 INTRDBRQS (on page 445)

**NAME**

MGRVLV — Manager Level

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1442'

**Length** \*

**Class** CLASS

**Sprcls** DATA - Encoded Information

Manager Level (MGRVLV) associates the codepoint of a class of managers with a server's level of support for that class. Level 0 specifies that the class of managers is not supported.

**Length Specification**

The length specified by the LENGTH attribute is 2 bytes.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>		<b>CLASS INSTANCE VARIABLES</b>
codpnt	INSTANCE_OF	CODPNTDR - Codepoint Data Representation
	NOTE	Specifies the codepoint of a manager class.
	ENUVAL	X'14CC' - CCSIDMGR - CCSID Manager
	MINLVL	4
	ENUVAL	X'1444' - CMNAPPC - LU 6.2 Conversational Communications Manager
	ENUVAL	X'147C' - CMNSYNCPT - SNA LU 6.2 Sync Point Conversational Communications Manager
	MINLVL	4
	ENUVAL	X'1474' - CMNTCPIP - TCP/IP Communication Manager
	MINLVL	5
	ENUVAL	X'1458' - DICTIONARY - Dictionary
	ENUVAL	X'240F' - RDB - Relational Database
	MINLVL	3
	ENUVAL	X'14C1' - RSYNCMGR - Resynchronization Manager
	MINLVL	5
	ENUVAL	X'1440' - SECMGR - Security Manager
	ENUVAL	X'2407' - SQLAM - SQL Application Manager
	MINLVL	3
	ENUVAL	X'143C' - SUPERVISOR - Supervisor
	ENUVAL	X'14C0' - SYNCPTMGR - Sync Point Manager
	MINLVL	4
	ENUVAL	X'1C01' - XAMGR - XA Manager
	NOTE	This must be treated as an open-ended list because of its use in the EXCSAT command. That is, codepoints other than those in the previously enumerated value list must be accepted and handled as the EXCSAT command requires.



REQUIRED		
<b>mgrlvln</b>	INSTANCE_OF	MGRLVLN - Manager-level Number Attribute
	SPCVAL	0
	NOTE	The <i>mgrlvln</i> parameter documented in each manager term determines the manager level number.
REQUIRED		
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

<b>insvar</b>	<i>MGRLVLLS</i> (on page 508) <i>MGRLVLOVR</i> (on page 511)
<b>Semantic</b>	<i>DCESECOVR</i> (on page 253) <i>LVLCMP</i> (on page 486) <i>SCALAR</i> (on page 796) <i>USRSECOVR</i> (on page 1050)

**NAME**

MGRVLVLS — Manager-level List

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1404'

**Length** \*

**Class** CLASS

**Sprcls** SCALAR - Scalar Object

Manager-level List (MGRVLVLS) Scalar Object specifies a list of codepoints and support levels for the classes of managers a server supports.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
length	*
class	X'1404'
mgrlvl	INSTANCE_OF MGRVLV - Manager Level REPEATABLE REQUIRED
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO**

- insvar**
  - EXCSAT* (on page 363)
  - EXCSATRD* (on page 369)
  - MGRVLVLRM* (on page 512)
- Semantic**
  - DCESECOVR* (on page 253)
  - LVLCMP* (on page 486)
  - MGRVLVLRM* (on page 512)
  - USRSECOVR* (on page 1050)

**NAME**

MGRLVLN — Manager-level Number Attribute

**DESCRIPTION (Semantic)****Dictionary** QDDBASD**Codepoint** X'1473'**Length** \***Class** CLASS**Sprcls** BIN - Binary Integer Number

Manager-level Number Attribute (MGRLVLN) Binary Integer Number specifies the level of a defined DDM manager. Any change to the function between a lower and higher level of the DDM architecture that a manager provides causes the level number of the manager to change to the higher level.

Manager levels distinguish between different, upward-compatible levels of a given file, access method, or other DDM manager. The EXCSAT command can then exchange information about what levels of each manager class the source and target systems support; therefore, level dependencies between managers can be specified and validated.

Each manager level is associated with a specific version of the architecture. Thus, Level 1 is associated with DDM Level 1, Level 2 with DDM Level 2, and so on. When a new manager is introduced into the architecture, it is given the architecture-level number in which it is introduced. Thus, the Stream File model was initially assigned manager Level 2 since it was introduced in DDM Level 2.

When any aspect of an existing manager is changed, it is assigned the architecture-level number in which it is changed. For example, in DDM Level 5, the ACCSEC command was introduced. Since this command adds to the instance commands list of the SECMGR manager, the manager-level number of SECMGR was updated to Level 5. If a manager is not changed in a new architecture level, it retains its current manager level.

Any of the following changes to a manager upgrades the level number of that manager:

1. Any changes to the manager class itself
2. Any change to any term referenced in the description of the manager class, such as the terms that define its instance variables, commands, command data, replies, reply data, and so on
3. Any change to indirectly referenced terms
4. Any change to other managers that have a semantic effect on other managers; for example, the addition of the CMNTCPIP manager in DDM Level 5 adds additional processing to the SYNCPTMGR manager, causing it to be incremented to Level 5 also

A table has been added to the manager class terms that shows the level by level changes to the manager and contains some of the information outlined above. The information in the table is independent of the OPTIONAL or REQUIRED nature of the changes. Footnotes are occasionally added to highlight important changes, such as addressing streams with offset instead of position. In most cases, several changes increase a manager's level. Implementors must choose which changes to build into their products. As long as one change is implemented, the manager level is increased to the DDM architecture level containing the functional description.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
<b>length</b>	*	
<b>class</b>	X'1473'	
<b>value</b>	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	16
	REQUIRED	
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

- mgrdepls**      *AGENT* (on page 61)  
                   *CMNSYNCPT* (on page 202)  
                   *MANAGER* (on page 491)  
                   *QDDBASD* (on page 651)  
                   *QDDRDBD* (on page 657)  
                   *RSYNCMGR* (on page 787)  
                   *SQLAM* (on page 847)  
                   *SYNCPTMGR* (on page 939)
- mgrlvln**        *MANAGER* (on page 491)  
                   *MGRLVL* (on page 506)

**NAME**

MGRLVLOVR — Manager Level Overrides

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD  
**Codepoint** X'1C03'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

Manager Level Overrides (MGRLVLOVR) specifies the manager levels that override those selected for use in the connection by the EXCSAT and EXCSATRD exchanged between a requester and a server. See the DRDA Reference for further information on the usage of the MGRLVLOVR parameter.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'1C03'	
<b>mgrlvl</b>	INSTANCE_OF REPEATABLE REQUIRED MINLVL	MGRLVL - Manager Level   7
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** PRCCNVCD (on page 621)  
**rpydta** BGNBND (on page 110)  
 BNDSQLSTT (on page 136)  
 CLSQRY (on page 165)  
 CNTQRY (on page 222)  
 DRPPKG (on page 293)  
 DSCRDBTBL (on page 300)  
 DSCSQLSTT (on page 304)  
 ENDBND (on page 336)  
 EXCSQLIMM (on page 371)  
 EXCSQLSET (on page 377)  
 EXCSQLSTT (on page 381)  
 OPNQRY (on page 555)  
 PRPSQLSTT (on page 636)  
 RDBCMM (on page 728)  
 RDBRLLBCK (on page 745)  
 REBIND (on page 753)  
**Semantic** RSLSETFLG (on page 783)  
 TYPSQLDA (on page 1034)

**NAME**

MGRVLVLRM — Manager-level Conflict

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1210'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

Manager-level Conflict (MGRVLVLRM) Reply Message indicates that the manager levels specified in the MGRVLVLLS conflict among themselves or with previously specified manager levels:

- The manager-level dependencies of one specified manager violates another specified manager level.
- The manager level specified attempts to respecify a manager level that a previous EXCSAT command specified.

**DSS Carrier:** RPYDSS

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'1210'	
<b>mgrlvl</b>	INSTANCE_OF REQUIRED NOTE	MGRVLVLLS - Manager-level List  The manager levels that the MGRVLVLLS parameter specifies on EXCSAT conflict.
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code  8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**cmdrpy** EXCSAT (on page 363)

**Semantic** EXCSAT (on page 363)

**NAME**

MGRNAM — Manager Name

**DESCRIPTION (Semantic)****Dictionary** QDDBASD**Codepoint** X'1452'**Length** \***Class** CLASS**Sprcls** NAME - Name

Manager Name (MGRNAM) is an unarchitected character string. In DDM architecture, a user provides a manager name to the DDM source server. This name must be in the format that the target server requires for creating or locating the manager.

It is up to the target agent to validate the name according to its rules. This can be done before or after attempting to use the specified manager name.

No semantic meaning is assigned to manager names in DDM.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'1452'	
value	INSTANCE_OF	BYTSTRDR - Byte String
	MINLEN	0
	MAXLEN	255
	REQUIRED	
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**Semantic** *PRCNAM* (on page 627)  
*RDBNAM* (on page 736)  
*SECMGRNM* (on page 818)  
*SPVNAM* (on page 834)  
*SRVNAM* (on page 880)

**vldattls** *CCSIDMGR* (on page 150)  
*CMNAPPC* (on page 184)  
*CMNMGR* (on page 196)  
*CMNSYNCPT* (on page 202)  
*CMNTCPIP* (on page 214)  
*DICTIONARY* (on page 286)  
*MANAGER* (on page 491)  
*RSYNCMGR* (on page 787)  
*SQLAM* (on page 847)  
*SYNCPTMGR* (on page 939)  
*XAMGR* (on page 1063)

## NAME

MGROVR — Manager Layer Overview

## DESCRIPTION (Semantic)

**Dictionary** QDDTTRD**Length** \***Class** HELP

Manager Layer Overview (MGROVR) discusses DDM managers and their purpose in the DDM architecture. DDM managers, such as files and relational databases, are self-identifying structures that are encapsulated within a server. A remote server can access a manager only through the Agents (and Communication Managers) of the encapsulating server. The objects encapsulated within a manager are accessible only through manager interfaces.

DDM managers use the services of their system to provide other services to remote requesters, whereas system services use DDM managers to provide remote services to their local requesters. This is illustrated in Figure 3-49 (on page 515). The goal of the DDM architecture is to provide local/remote transparency for both local and remote requesters and services.

Within a server, DDM managers interact with each other to provide specific services as illustrated in Figure 3-50 (on page 515) and Figure 3-51 (on page 516). Two types of interactions occur—simple interactions and bound interactions. A simple interaction occurs when one manager requests a specific service from another manager, and the service is immediately provided. For example, an access manager uses the directory services to locate a file. The directory locates the file and returns it to the access manager. In providing its service, the directory uses the services of remote directories or copies of distributed directories. Bound interactions occur when managers are bound together to support a series of interactions over a period of time. For example, an access manager is bound to a file to support a series of application program requests for interactions with the file.

DDM services are viewed as local services, with all interactions between managers apparently occurring on the local system. If remote services are needed, then the manager providing the service is broken into two parts—a source server requesting remote services and a target server providing remote services. DDM Agents and Communication Managers act as intermediaries enabling communications between source server and target server.

Managers are described by CLASS objects stored in DDM managers called DICTIONARIES. Manager classes are described as a hierarchy (through ranked order) so that they can inherit interfaces (commands) and internal structures from more abstract manager classes.

Since the class object of a DDM manager describes the structure of its instances, it therefore defines their interchange data stream form. For example, it is possible to transmit an entire DDM keyed file, including all attributes, descriptors, and data from one server to another server.

See the descriptions of the following terms for information about DDM managers:

CCSIDMGR	CCSID Manager (see <i>CCSIDMGR</i> (on page 150))
CMNOVR	Communications Overview (see <i>CMNOVR</i> (on page 201))
RDBOVR	Relational Database Overview (see <i>RDBOVR</i> (on page 740))



A System

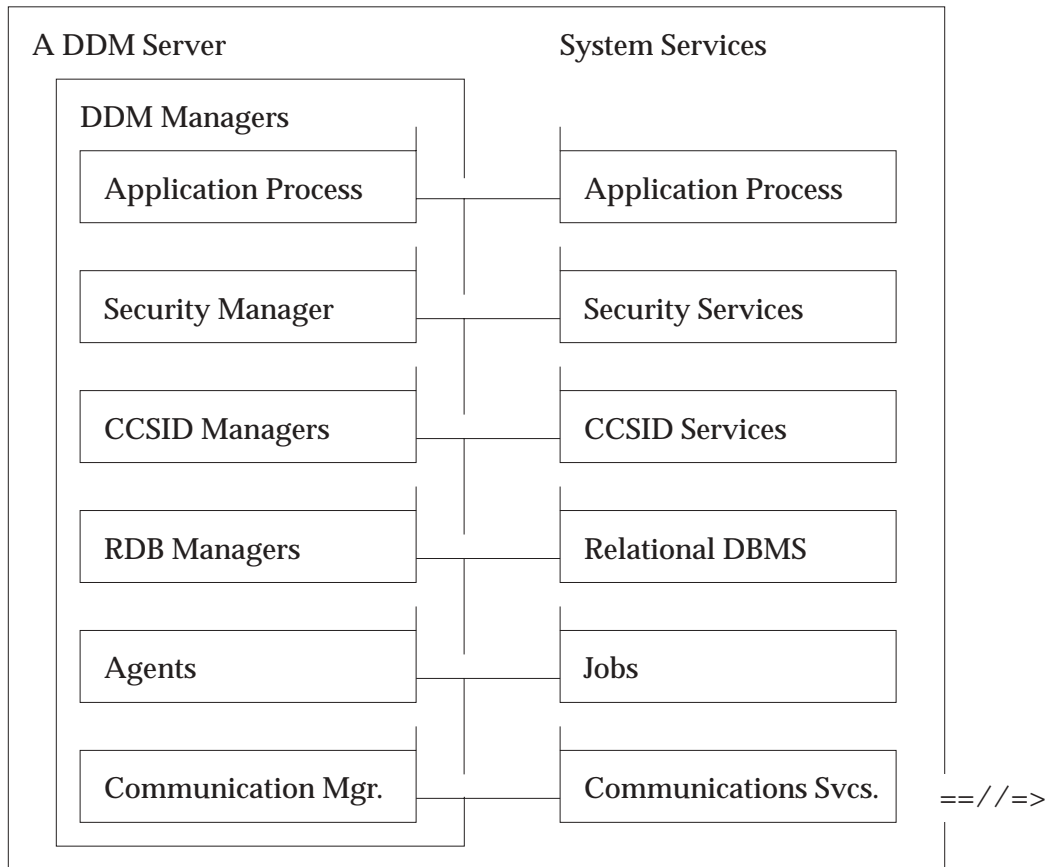


Figure 3-49 Mapping a DDM Server onto a System

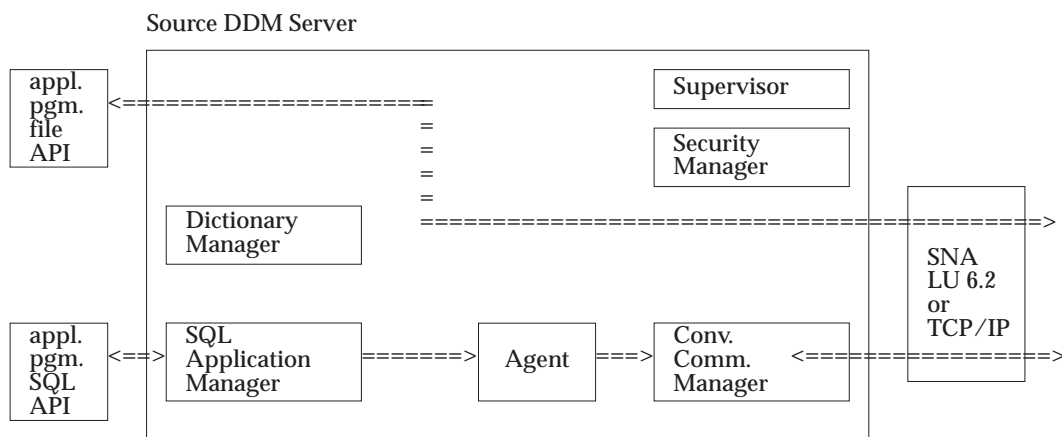


Figure 3-50 Source Server Manager Interactions

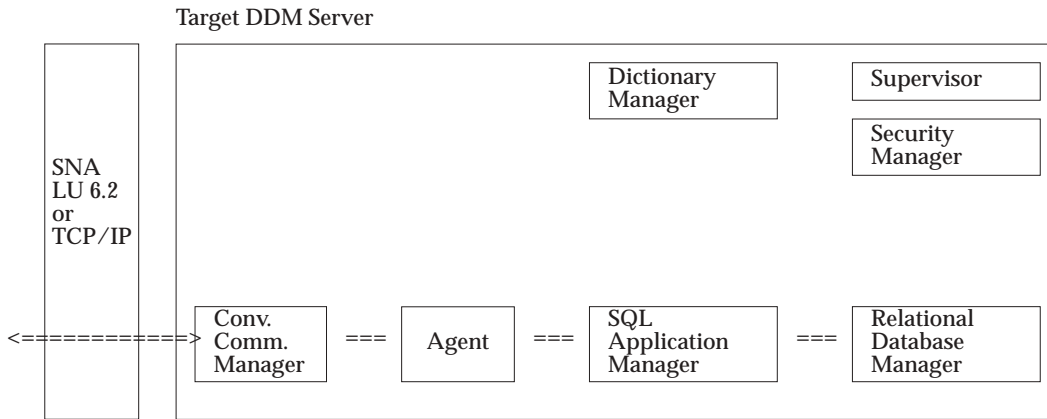


Figure 3-51 Target Server Manager Interactions

SEE ALSO

Semantic

DDM (on page 260)

**NAME**

MINLEN — Minimum Length Attribute

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD  
**Codepoint** X'0025'  
**Length** \*  
**Class** CLASS  
**Sprcls** BIN - Binary Integer Number

Minimum Length Attribute (MINLEN) Binary Integer Number specifies that the attribute value is the minimum length of a term.

The unit of measurement varies according to the class of the term being described. For example, the unit of a bit string (BITSTRDR) is a bit (BITDR), while the unit of a character string (CHRSTRDR) is a character (CHRDR).

<b>clsvar</b>	NIL
<b>insvar</b>	CLASS INSTANCE VARIABLES
<b>length</b>	*
<b>class</b>	X'0025'
<b>value</b>	INSTANCE_OF BINDR - Binary Number Field REQUIRED
<b>clscmd</b>	NIL
<b>inscmd</b>	NIL

**SEE ALSO**

**insvar** *BNDOPTNM* (on page 134)  
*BNDOPTVL* (on page 135)  
*CRRTKN* (on page 246)  
*MGRNAM* (on page 513)  
*NEWPASSWORD* (on page 531)  
*PASSWORD* (on page 578)  
*PKGOWNID* (on page 600)  
*RPYDSS* (on page 766)  
*RQSDSS* (on page 774)  
*TEXT* (on page 1018)  
*TITLE* (on page 1020)  
*USRID* (on page 1044)

**NAME**

MINLVL — Minimum Level Number Attribute

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'0002'

**Length** \*

**Class** CLASS

**Sprcls** BIN - Binary Integer Number

Minimum Level Number specifies the minimum DDM architecture level required to use an object. Using an object with a manager-level number less than specified results in compatibility errors.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'0002'	
value	INSTANCE_OF LENGTH REQUIRED	BINDR - Binary Number Field 16
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

None.

**NAME**

MINVAL — Minimum Value Attribute

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD  
**Codepoint** X'0026'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

Minimum Value Attribute (MINVAL) Scalar Object specifies that the attribute value is the minimum valid value for a term.

The attribute object specified must have attributes compatible with those specified for the term.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'0026'	
<b>value</b>	SPRCLS NOTE	FIELD - A Discrete Unit of Data The attributes for this variable vary according to what is being defined. Also, when BITDR, BITSTRDR, HEXDR, or HEXSTRDR values are specified, they are represented as left-justified in the fewest number of whole bytes possible. For example, 123 would be X'1230'.
	REQUIRED	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** AGNPRMRM (on page 65)  
 CMDCHKRM (on page 173)  
 DGRIOPRL (on page 279)  
 DSCERRCD (on page 296)  
 FDODSCOFF (on page 407)  
 FDODTAOFF (on page 409)  
 FDOPRMOFF (on page 413)  
 FDOTRPOFF (on page 414)  
 MAXBLKEXT (on page 494)  
 MAXRSLCNT (on page 497)  
 MAXSCTNBR (on page 498)  
 OBJDSS (on page 536)  
 OBJNSPRM (on page 542)  
 PKGSN (on page 606)  
 PRCCNVRM (on page 625)  
 QRYBLKSZ (on page 678)  
 RPYDSS (on page 766)  
 RQSCRRL (on page 772)

*RQSDSS* (on page 774)  
*RSCLMTRM* (on page 778)  
*SQLSTTNBR* (on page 870)  
*SYNTAXRM* (on page 989)  
*VALNSPRM* (on page 1057)

**NAME**

MONITOR — Monitor Events

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1900'

**Length** \*

**Class** CLASS

**Sprcls** BITSTRDR - A Bit String Field

Specify a set of events or activities to be monitored on the target server when processing the command. MONITOR allows the definition of up to 32 different events that can be specified on a command and monitored by the target server. Each event or activity to be monitored is defined as an ordered and contiguous set of a bits where each bit is a boolean flag.

**Supported Events**

The default (X'0000 0000') bit string means that no monitoring is requested for this command and is the default value.

The *etime* (X'8000 0000') flag requests the *server elapsed CPU time*. The source server requests the target server to return the ETIME in the MONITORRD object. The ETIME contains the CPU time to parse the request data stream, process the command, and generate the reply data stream. No network time to receive or send the data stream should be included in the ETIME. The target server is not required to support the monitoring of the ETIME at the server. If not supported, the request is ignored and MONITORRD is not returned.

<b>clsvar</b>	NIL	
<b>insvar</b>	INSTANCE VARIABLES	
length	8	
class	X'1900'	
etime	INSTANCE_OF	BITDR - A Single Bit
	IGNORABLE	
	ENUVAL	B'1' - Return server ETIME.
	ENUVAL	B'0' - Do not return server ETIME.
reserved	DFTVAL	B'0' - Do not return server ETIME
	INSTANCE_OF	BITDR - A Single Bit
	LENGTH	31
	DFTVAL	B'000....0000'
RESERVED		
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

## SEE ALSO

- |                 |   |
|-----------------|---|
| <b>insvar</b>   | <i>CNTQRY</i> (on page 222)<br><i>EXCSQLSTT</i> (on page 381)<br><i>OPNQRY</i> (on page 555)  |
| <b>Semantic</b> | <i>CLSQRY</i> (on page 165)<br><i>CNTQRY</i> (on page 222)<br><i>EXCSQLSTT</i> (on page 381)<br><i>MONITORRD</i> (on page 523)<br><i>OPNQRY</i> (on page 555) |



**NAME**

MONITORRD — Monitor Reply Data

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD  
**Codepoint** X'1C00'  
**Length** \*  
**Class** CLASS  
**Sprcls** COLLECTION - Collection Object

The MONITOR Reply Data (MONITORRD) allows the target agent to return monitoring data to the source agent. The MONITORRD is generated as a separate object and must be the last object in the reply for the request.

**Source System Processing**

If the source agent specifies the MONITOR instance variable on a command requesting an event or activity to be monitored, The specific event or activity specified identifies whether monitoring data is to be returned in the reply to the command. Currently ETIME is the only monitor type defined. Refer to ETIME for the specific reply data requirements. If the target server does not support the event or activity requested, the MONITORRD is not required to be sent.

**Target System Processing**

The target agent monitors the requested event or activity and returns a MONITORRD if required by the type of event or activity monitored. Refer to *MONITOR* (on page 521) for the list of supported events and activities and their reply data requirements. The MONITORRD must be the last object in the chain of data stream objects generated for the request.

**DSS Carrier: OBJDSS**

<b>clsvar</b>	NIL
<b>insvar</b>	INSTANCE VARIABLES
<b>length</b>	*
<b>class</b>	X'1C00'
<b>etime</b>	INSTANCE_OF ETIME - Elapsed CPU Time OPTIONAL
<b>clscmd</b>	NIL
<b>inscmd</b>	NIL

**SEE ALSO**

**rpydta** CNTQRY (on page 222)  
 OPNQRY (on page 555)  
**Semantic** MONITOR (on page 521)

**NAME**

MTLEXC — Mutually-exclusive Attribute

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'0067'

**Length** \*

**Class** CLASS

**Sprcls** OBJECT - Self-identifying Data

Mutually-exclusive Attribute (MLTEXC) identifies by codepoint an object that cannot exist in the same collection as the object with the MTLEXC attribute.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
length	*
class	X'0067'
value	INSTANCE_OF CODPNTDR - Codepoint Data Representation REQUIRED
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO**

- insvar** *SRVLSRV* (on page 876)  
*SYNCLOG* (on page 922)
- rpydta** *DSCRDBTBL* (on page 300)  
*DSCSQLSTT* (on page 304)  
*EXCSQLSTT* (on page 381)  
*PRPSQLSTT* (on page 636)
- Semantic** *CNTQRY* (on page 222)  
*LVLCMP* (on page 486)  
*REQUIRED* (on page 761)

**NAME**

MTLINC — Mutually-inclusive Attribute

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'00A7'

**Length** \*

**Class** CLASS

**Sprcls** OBJECT - Self-identifying Data

Mutually-inclusive Attribute (MTLINC) identifies by codepoint an object that must exist in the same collection as the object with the MTLINC attribute.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
length	*
class	X'00A7'
value	INSTANCE_OF CODPNTDR - Codepoint Data Representation REQUIRED
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO**

**rpydta** ACCRDB (on page 42)

**NAME**

NAMDR — Name Data

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'0066'

**Length** \*

**Class** CLASS

**Sprcls** FIELD - A Discrete Unit of Data

The Name Data (NAMDR) field specifies the name of a resource or of a DDM term.

**Length Specification**

The length of a NAMDR field is specified in bytes.

**Literal Form**

Name literals are specified as quoted or unquoted character strings. Enclosing quotes are required when embedded blanks are in the name. Examples of name literals are *NAME*, *PAYROLL*, and *MESSAGE PROFILE*.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
value	INSTANCE_OF	CHRSTRDR - Character String
	MAXLEN	255
	REQUIRED	
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>vldattls</b>	<b>NIL</b>	

**SEE ALSO**

- insvar** *EXTNAM* (on page 402)
- PKGRPLVRS* (on page 605)
- PRCNAM* (on page 627)
- SRVNAM* (on page 880)
- TYPDEFNAM* (on page 1027)
- TYPFMLNM* (on page 1033)
- VRSNAM* (on page 1059)

**Semantic** *INHERITANCE* (on page 437)

NAME

NAME — Name

DESCRIPTION (Semantic)

**Dictionary** QDDPRMD

**Codepoint** X'0027'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Name (NAME) String specifies a name associated with an object.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'0027'	
value	INSTANCE_OF	CHRSTRDR - Character String
	MAXLEN	255
	OPTIONAL	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

SEE ALSO

- insvar**
  - DCTINDEN* (on page 259)
  - DEFINITION* (on page 267)
- Semantic**
  - DEFLST* (on page 268)
  - DFTRDBCOL* (on page 274)
  - DICTIONARY* (on page 286)
  - EXTNAM* (on page 402)
  - INHERITANCE* (on page 437)
  - MGRNAM* (on page 513)
  - NEWPASSWORD* (on page 531)
  - PASSWORD* (on page 578)
  - PKGOWNID* (on page 600)
  - PKGRPLVRS* (on page 605)
  - RSYNCTYP* (on page 790)
  - SRVCLSNM* (on page 872)
  - TYPDEFNAM* (on page 1027)
  - TYPFMLNM* (on page 1033)
  - USRID* (on page 1044)
  - VRSNAM* (on page 1059)

**NAME**

NAMSYMDR — Name Symbol Data Representation

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'0061'

**Length** \*

**Class** CLASS

**Sprcls** FIELD - A Discrete Unit of Data

Name Symbol Data Representation (NAMSYMDR) Field specifies a name containing only the characters: uppercase A through Z, character digits 0 through 9, and the underscore character. Trailing blanks are permissible, but blanks cannot appear anywhere else.

**Length Specifications**

The length of a NAMSYMDR field is specified in bytes.

clsvar	NIL	
insvar	CLASS INSTANCE VARIABLES	
value	INSTANCE_OF	CHRDR - A Graphic Character
	ENUVAL	'A'
	ENUVAL	'B'
	ENUVAL	'C'
	ENUVAL	'D'
	ENUVAL	'E'
	ENUVAL	'F'
	ENUVAL	'G'
	ENUVAL	'H'
	ENUVAL	'I'
	ENUVAL	'J'
	ENUVAL	'K'
	ENUVAL	'L'
	ENUVAL	'M'
	ENUVAL	'N'
	ENUVAL	'O'
	ENUVAL	'P'
	ENUVAL	'Q'
	ENUVAL	'R'
	ENUVAL	'S'
	ENUVAL	'T'
	ENUVAL	'U'
	ENUVAL	'V'
	ENUVAL	'W'
	ENUVAL	'X'
	ENUVAL	'Y'
	ENUVAL	'Z'

	ENUVAL	'0'
	ENUVAL	'1'
	ENUVAL	'2'
	ENUVAL	'3'
	ENUVAL	'4'
	ENUVAL	'5'
	ENUVAL	'6'
	ENUVAL	'7'
	ENUVAL	'8'
	ENUVAL	'9'
	ENUVAL	' '
	ENUVAL	'_'
	ENUVAL	' '
	REPEATABLE	
	MAXLEN	255
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>vldattls</b>	<b>NIL</b>	

**SEE ALSO**

None.

**NAME**

NBRROW — Number of Input Rows

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'213A'  
**Length** \*  
**Class** CLASS  
**Sprcls** BIN - Binary Integer Number

Number of Input Rows (NBRROW) specifies the number of rows to input for multi-row input.

For a multi-row input operation, any null row counts towards the total number of rows for a multi-row input operation as indicated by NBRROW.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	8	
class	X'213A'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	32
	NOTE	For multi-row input, values of 1 ... N input up to N rows of data.
	MINVAL	1
	NOTE	For multi-row input, a value of zero is not valid.
	DFTVAL	1
	OPTIONAL	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** CNTQRY (on page 222)  
 EXCSQLSTT (on page 381)  
 QRYROWSET (on page 697)  
**Semantic** QRYROWSET (on page 697)



**NAME**

NEWPASSWORD — New Password

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'11DE'

**Length** \*

**Class** CLASS

**Sprcls** NAME - Name

NEWPASSWORD at the Target System specifies the new password associated with the defined end-user name.

With certain SECMECs, the new password can be encrypted.

The target server can optionally *fold* the new password if it is not in the correct form for the target security manager. Folding is the process of modifying the characters from one form to another. For instance, from mixed case into all upper or all lowercase.

The DDM architecture treats the NEWPASSWORD as a byte string. The SECMGR is aware of and has knowledge of the contents of the NEWPASSWORD. DDM does not know or care about the specific contents of this field. If the reader needs to know the actual contents, see *USRSECOVR* (on page 1050).

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'11DE'	
value	INSTANCE_OF	CHRSTRDR - Character String Data Representation
	MINLEN	1
	MAXLEN	255
	REQUIRED	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *SECCHK* (on page 800)

**Semantic** *SECCHK* (on page 800)  
*USRSECOVR* (on page 1050)

**NAME**

NIL — Nil Object

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'002A'

**Length** \*

**Class** CLASS

**Sprcls** NIL

NIL Object (NIL) is the class of all instances of NIL that serves as a means of identifying the class of NIL. NIL is:

- The default object used when no other object is valid.
- Not a subclass of OBJECT.
- The default object associated with variables not initialized.
- Often used as the terminator of lists or hierarchies of objects.
- The object returned when no other object meets indicated selection or evaluation criteria. When NIL is specified for an instance variable of a collection contained within a SPACE, and the SPCPTR attribute is also specified, then 00000000 is the encoded value of the instance variable.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
<b>length</b>	<b>4</b>
<b>class</b>	<b>X'002A'</b>
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO**

None.

**NAME**

NOTE — Note Attribute

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'0014'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Note Attribute (NOTE) String specifies text that describes or qualifies the previous attributes in a definition.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
length	*
class	X'0014'
value	INSTANCE_OF REQUIRED CHRSTRDR - Character String
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO**

None.

## NAME

NUMBER — Number

## DESCRIPTION (Semantic)

**Dictionary** QDDPRMD**Codepoint** X'002B'**Length** \***Class** CLASS**Sprcls** MAGNITUDE - Linearly Comparable Scalar

Number (NUMBER) specifies the general protocol applicable to all classes of numbers.

---

**clsvar** NIL

---

**insvar** NIL

---

**clscmd** NIL

---

**inscmd** NIL

## SEE ALSO

**Semantic** *BIN* (on page 118)  
*INHERITANCE* (on page 437)  
*OBJOVR* (on page 544)  
*OOPOVR* (on page 547)

**NAME**

NWPWDSEC — New Password Security Mechanism

**DESCRIPTION (Semantic)**

**Dictionary** QDDTTRD

**Length** \*

**Class** HELP

The New Password Security Mechanism (NWPWDSEC) specifies the use of the new password security mechanism. See *USRSECOVR* (on page 1050) for more information about the new password security mechanism.

**SEE ALSO**

**Semantic**        *SECMEC* (on page 811)  
                  *USRENCPWD* (on page 1043)  
                  *USRIDNWPWD* (on page 1045)

**NAME**

OBJDSS — Object Data Stream Structure

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1429'

**Length** \*

**Class** CLASS

**Sprcls** DSS - Data Stream Structures

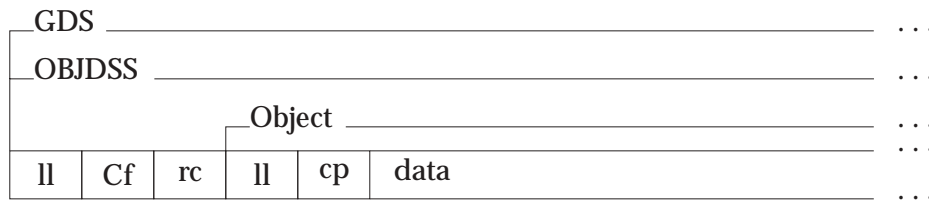
Object Data Stream Structure (OBJDSS) specifies the general format of DDM object data stream structures.

These DSSs carry all objects except commands and reply messages. In particular, they carry records, file attributes, and feedback information.

The number of objects carried in a single OBJDSS depends on the transformations the communications manager performs. One or more objects can be carried. The communications managers must deliver normalized objects to the target agent.

The objects can be encrypted.

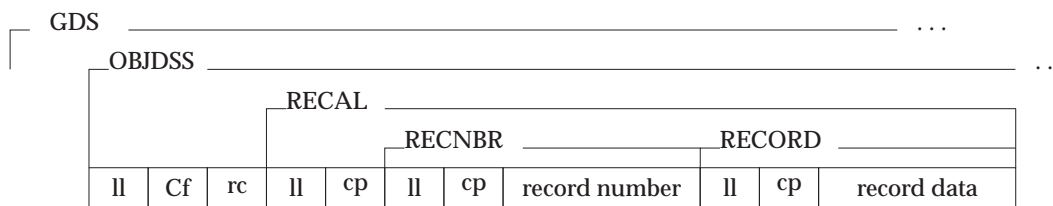
All fields of the OBJDSS header must be specified in the order shown in Figure 3-52 through Figure 3-54 (on page 537) because they are not self-defining structures.



**Figure 3-52** Data Objects

**Legend**

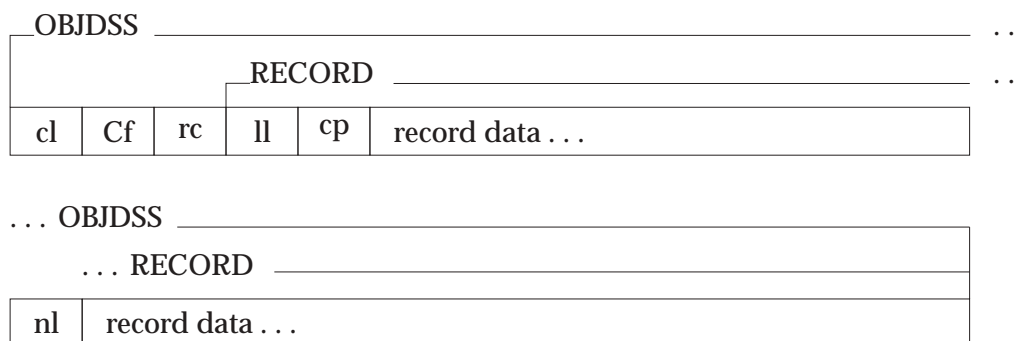
- ll Two-byte length field.
- C D0
- f One-byte format = object data stream structure.
- rc Two-byte request correlation identifier.
- data Object data value.



**Figure 3-53** A Single Record with Record Number Feedback

**Legend**

- ll Two-byte length field.
- C D0
- f One-byte format = object data stream structure.
- rc Two-byte request correlation identifier.
- cp Two-byte object class codepoint (RECAL, RECNR, or RECORD).
- data Object data value.



**Figure 3-54** A Single Record Formatted in a Continued OBJDSS

**Legend**

- cl Two-byte length field, high-order bit set to B'1'.
- nl Two-byte length field, high-order bit set to B'0'.
- C D0
- f One-byte format = object data stream structure.
- rc Two-byte request correlation identifier.
- cp Two-byte object class codepoint (of RECORD).

clsvar	NIL		
insvar	CLASS INSTANCE VARIABLES		
length	INSTANCE_OF	BINDR - Binary Number Field	
	LENGTH	16	
	MINVAL	6	
	MAXVAL	32,767	
	NOTE	Specifies the length of an OBJDSS, including the length field. If it is necessary to include more data than the maximum allows, or if it is desirable to break a large structure into smaller pieces, set the high-order bit of the length field to B'1'. This indicates that the structure is continued in the next structure transmitted.	
		REQUIRED	
	ddmid	INSTANCE_OF REQUIRED	DDMID - DDM Identifier
	format	INSTANCE_OF	DSSFMT - Data Stream Structure Format
		ENUVAL	03
		NOTE	Unchained OBJDSS.
ENUVAL		43	
NOTE		Chained OBJDSS and the next DSS have different request correlator (RQSCRR).	
ENUVAL		53	
NOTE		Chained OBJDSS and the next DSS have the same request correlator (RQSCRR).	
ENUVAL		04	
NOTE		Unchained encrypted OBJDSS.	
MINLVL		5	
ENUVAL	44		
NOTE	Chained encrypted OBJDSS and the next DSS have different request correlator (RQSCRR).		
MINLVL	5		
ENUVAL	54		
NOTE	Chained encrypted OBJDSS and the next DSS have the same request correlator (RQSCRR).		
MINLVL	5		
REQUIRED			
rqscrr	INSTANCE_OF	RQSCRR - Request Correlation Identifier	
	NOTE	Specifies a request identifier the source communications manager assigns. This identifier is the same for all DSSs associated with a request, including the RQSDSS and any OBJDSSs sent with it. The same identifier is also specified for all DSSs in response to the RQSDSS, including RPYDSSs and any reply OBJDSSs.	
	REQUIRED		



data	INSTANCE_OF NOTE	BYTSTRDR - Byte String The contents of this field are determined by the data object to be transmitted.
	REQUIRED	
clscmd	NIL	
inscmd	NIL	

## SEE ALSO

<b>cmdtda</b>	<i>COMMAND</i> (on page 240)
<b>insvar</b>	<i>PRCCNVCD</i> (on page 621)
<b>rpydta</b>	<i>COMMAND</i> (on page 240)
<b>Semantic</b>	<i>AGNCMDPR</i> (on page 64) <i>APPCMNI</i> (on page 72) <i>APPSRCCD</i> (on page 79) <i>APPSRCCR</i> (on page 86) <i>APPSRCER</i> (on page 91) <i>APPTRGER</i> (on page 95) <i>CMNAPPC</i> (on page 184) <i>CMNMGR</i> (on page 196) <i>CMNSYNCPT</i> (on page 202) <i>CMNTCPIP</i> (on page 214) <i>COMMAND</i> (on page 240) <i>DSS</i> (on page 308) <i>EDTASECOVR</i> (on page 323) <i>EUSRIDDTA</i> (on page 355) <i>EUSRNPWDDTA</i> (on page 359) <i>EUSRPWDDTA</i> (on page 361) <i>EXCSATRD</i> (on page 369) <i>FDOOBJ</i> (on page 411) <i>INHERITANCE</i> (on page 437) <i>LMTBLKPRC</i> (on page 475) <i>OBINSPRM</i> (on page 542) <i>QRYBLK</i> (on page 670) <i>RDBOVR</i> (on page 740) <i>RPYDSS</i> (on page 766) <i>RQSCRR</i> (on page 772) <i>RQSDSS</i> (on page 774) <i>SQLDTA</i> (on page 860) <i>SQLDTARD</i> (on page 862) <i>SQLSTT</i> (on page 868) <i>SYNCCRD</i> (on page 913) <i>SYNCLOG</i> (on page 922) <i>SYNCMNI</i> (on page 931) <i>SYNCRRD</i> (on page 981) <i>TCPCMNI</i> (on page 994) <i>TCPSRCCD</i> (on page 1007) <i>TCPSRCCR</i> (on page 1010) <i>TYPDEF</i> (on page 1025)

## NAME

OBJECT — Self-identifying Data

## DESCRIPTION (Semantic)

**Dictionary** QDDPRMD

**Codepoint** X'002C'

**Length** \*

**Class** CLASS

**Sprcls** DATA - Encoded Information

Object (OBJECT) is a data processing entity that structures and stores data and exhibits well-defined behaviors in response to specific commands.

All objects the DDM architecture recognizes are instances of a CLASS object that DDM defines. A CLASS object defines the variables of its instances with its *insvar* list.

The DDM OBJECT class has four primary subclasses:

- SCALAR objects consist of zero or more instances of the FIELD subclasses of DDM. If more than one instance variable is defined, then they are mapped into the value of the object in the order which they are defined.
- SPACES are special cases of SCALAR objects consisting of a single instance variable whose value is a space in which other objects can be encoded.
- COLLECTION objects consist of zero or more instances of the SCALAR and COLLECTION subclasses of DDM.
- MANAGER objects manage a collection of objects.

The CLASS description of a collection's instance variable is independent of the collection stored in a space or encoded in a data stream for transmission between systems. See SPACE for a description of how collections are encoded in spaces. And see *DSS* (on page 308) for a description of how collections are encoded for transmission between systems.

A CLASS object specifies the commands its instances respond to with its instance commands (*inscmd*) list. DDM does not provide a complete definition for the classes of objects defined or used within DDM. Commands are defined for the major data management functions which require architected interfaces, such as files and access methods. Commands are not defined for other classes, such as binary numbers or character strings, for internal interfaces between components, or for the object-oriented commands (messages or member functions) such as *SELF*.

**Length Specification**

The LENGTH attribute specifies the length in bytes.

**Literal Form**

The literal form of all objects is:

`object_class_name(object_value)`

For example, the literal form of a file's record is:

`RECORD(value of the record)`

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
<b>length</b>	INSTANCE_OF LENGTH NOTE	BINDR - Binary Number Field 16 The length of an object includes its own length and the length of all subsequent variables in the object (in bytes).
	REQUIRED	
<b>class</b>	INSTANCE_OF NOTE REQUIRED	CODPNTDR - Codepoint Data Representation Specifies the codepoint of the class of the object.
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

- insvar**
  - ARRAY* (on page 101)
  - ASSOCIATION* (on page 103)
  - ATTLST* (on page 105)
  - QLFATT* (on page 664)
- Semantic**
  - CLASS* (on page 158)
  - CMDTRG* (on page 179)
  - COLLECTION* (on page 238)
  - DSS* (on page 308)
  - IGNORABLE* (on page 435)
  - INHERITANCE* (on page 437)
  - INHERITED* (on page 442)
  - LMTBLKPRC* (on page 475)
  - MANAGER* (on page 491)
  - MTLEXC* (on page 524)
  - MTLINC* (on page 525)
  - NIL* (on page 532)
  - OBJOVR* (on page 544)
  - OOPOVR* (on page 547)
  - OPTIONAL* (on page 568)
  - REPEATABLE* (on page 758)
  - REQUIRED* (on page 761)
  - SCALAR* (on page 796)
  - STRLYR* (on page 890)
  - SUBSETS* (on page 902)

**NAME**

OBJNSPRM — Object Not Supported

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD  
**Codepoint** X'1253'  
**Length** \*  
**Class** CLASS  
**Sprcls** RPYMSG - Reply Message

Object Not Supported (OBJNSPRM) Reply Message indicates that the target server does not recognize or support the object specified as data in an OBJDSS for the command associated with the object.

The OBJNSPRM is also returned if an object is found in a valid collection in an OBJDSS (such as the RECAL collection) that is not valid for that collection.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'1253'	
<b>codpnt</b>	INSTANCE_OF REQUIRED NOTE	CODPNT - Codepoint  This is the codepoint of the object that is not supported.
<b>rdbnam</b>	INSTANCE_OF MINLVL OPTIONAL	RDBNAM - Relational Database Name 3
<b>reccnt</b>	INSTANCE_OF MINVAL OPTIONAL NOTE  NOTE	RECCNT - Record Count 0  Required for requests to insert multiple records in a file.  This parameter is not returned by commands that operate on RDBs.
<b>svrdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL ENUVAL	SVRCOD - Severity Code  8 - ERROR - Error Severity Code 16 - SEVERE - Severe Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**cmdrpy**

*BGNBND* (on page 110)  
*BNDSQLSTT* (on page 136)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSET* (on page 377)  
*EXCSQLSTT* (on page 381)  
*OPNQRY* (on page 555)  
*PRPSQLSTT* (on page 636)  
*REBIND* (on page 753)  
*SECCHK* (on page 800)

**Semantic**

*CNTQRY* (on page 222)  
*EXCSQLSTT* (on page 381)  
*SUBSETS* (on page 902)

## NAME

OBJOVR — Object Layer Overview

## DESCRIPTION (Semantic)

**Dictionary** QDDTTRD**Length** \***Class** HELP

Object Layer Overview (OBJOVR) discusses the three kinds of DDM objects, as shown in Figure 3-55 (on page 545):

- Simple scalars contain only a single instance of one of the DDM data classes, such as a single number or a single character string. DDM attributes, such as LENGTH, ALIGNMENT, and SCALE are simple scalars.
- Mapped scalars contain a sequence of DDM data class instances that are mapped onto a byte stream by an external descriptor that specifies their class identifier and other attributes. The external descriptors can be stored along with the records of a file, or they can be stored in a dictionary for many files to use.
- Collections contain a sequence of scalar and collection objects. DDM commands, reply messages, and attribute lists are all examples of collection objects.

CLASS objects stored in the DDM DICTIONARY manager describe all objects. Object classes are described as a hierarchy so that they can inherit interfaces (commands) and variables from more abstract object classes. For example, the DDM LENGTH object class inherits from the BIN object class, and it inherits from the NUMBER object class.

Since the class of a DDM object describes its instances variables, it therefore defines their interchange data stream form, as shown in Figure 3-56 (on page 546). The interchange data stream form makes transmitting a command with multiple scalar parameters from one manager to another manager in a different server possible.

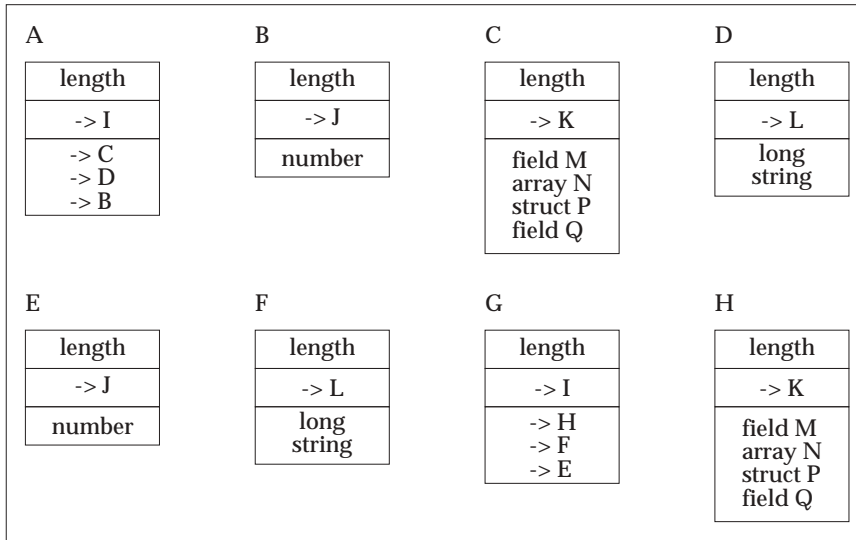
See the following terms for more information about objects:

**COLLECTION** Collection Object (see *COLLECTION* (on page 238))

**OBJECT** Self-identifying Data (see *OBJECT* (on page 540))

**SCALAR** Scalar Object (see *SCALAR* (on page 796))

A Manager Memory Heap



DDM DICTIONARY Manager

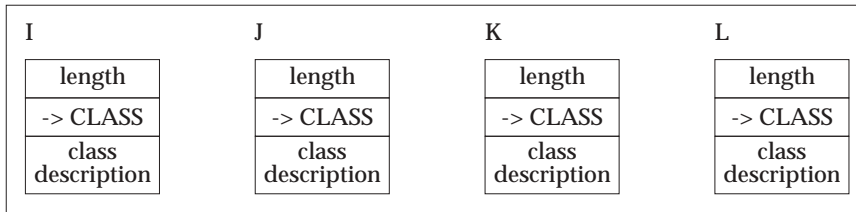


Figure 3-55 DDM Objects

Interchange Data Stream Form of Object A

length of A	class I codepoint
-------------	-------------------

NOTE: Length of A includes total length of the data stream representing A and all of its component objects.

length of C	class K codepoint	field M	array N	structure P	field Q
-------------	-------------------	---------	---------	-------------	---------

length of D	class L codepoint	long string
-------------	-------------------	-------------

length of B	class J codepoint	number
-------------	-------------------	--------

**Figure 3-56** DDM Object Interchange Format

**SEE ALSO**

**Semantic**      *DDM* (on page 260)



**NAME**

OOPOVR — Overview of Object-Oriented Programming using DDM

**DESCRIPTION (Semantic)**

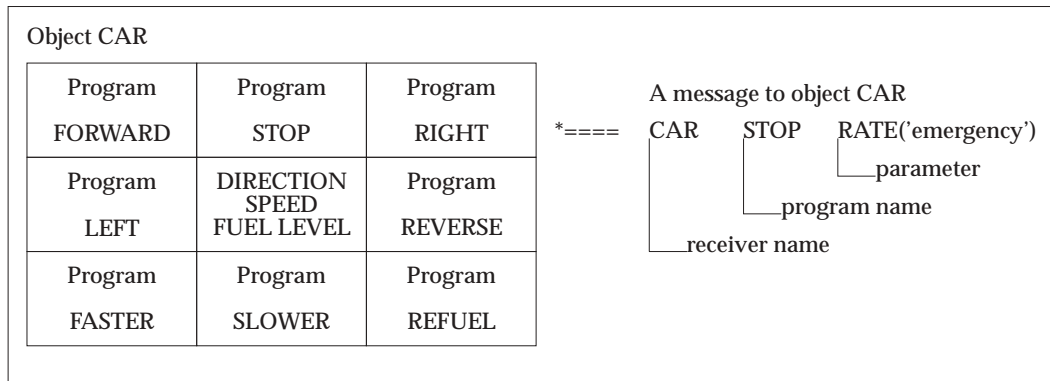
**Dictionary** QDDTTRD

**Length** \*

**Class** HELP

Object-Oriented Programming (OOP) perceives programming, communications, data management, and user interfaces such that it greatly enhances programmer productivity. It especially encourages the sharing and reusing of programs as off-the-shelf components. Instead of perceiving programming as a writing procedure, OOP emphasizes the definition of objects that model the application domain. For example, an application that models a highway system might include objects representing roads, traffic signals, and a variety of vehicles.

An object consists of data and the programs that operate on the data. Figure 3-57 illustrates this concept. A set of programs encapsulate the object's data. Only these programs can access or change the object's data. Sending a message to the object invokes these programs. The object to which a message is sent is called the receiver of the message. The message identifies a program and provides parameters to it. If the receiver has the specified program, the program is called and is passed the parameters of the message. To perform its function, the called program can access or modify the data values of the object and it can send messages to other objects. If the receiving object does not include the specified program, it responds that it does not understand the message.



**Figure 3-57** Objects—Data Items Encapsulated by Programs

The message sent to object CAR is processed by calling the program STOP and passing it the specified parameter. Only the programs of the object can modify the encapsulated variables of the object (DIRECTION, SPEED, and FUEL LEVEL).

In OOP, every object appears to have its own set of encapsulating programs. But of course, there are usually many instances of the same kind of object in an application. For example, there are many cars in a model of a highway system. Since all of them need the same set of programs, objects of the same kind share programs. Similarly, the data of all these objects consist of the same variables (scalar values or pointers to other objects), and the descriptions of these variables can also be shared.

Classes share both variable declarations and programs among a set of objects of the same kind. Instead of actually containing its programs and variable declarations, each object of the class

points to an object that describes the class, as shown in Figure 3-58. Then, when a message is sent to an object, one of the programs its class defines is invoked to access or modify the variables of the receiver.

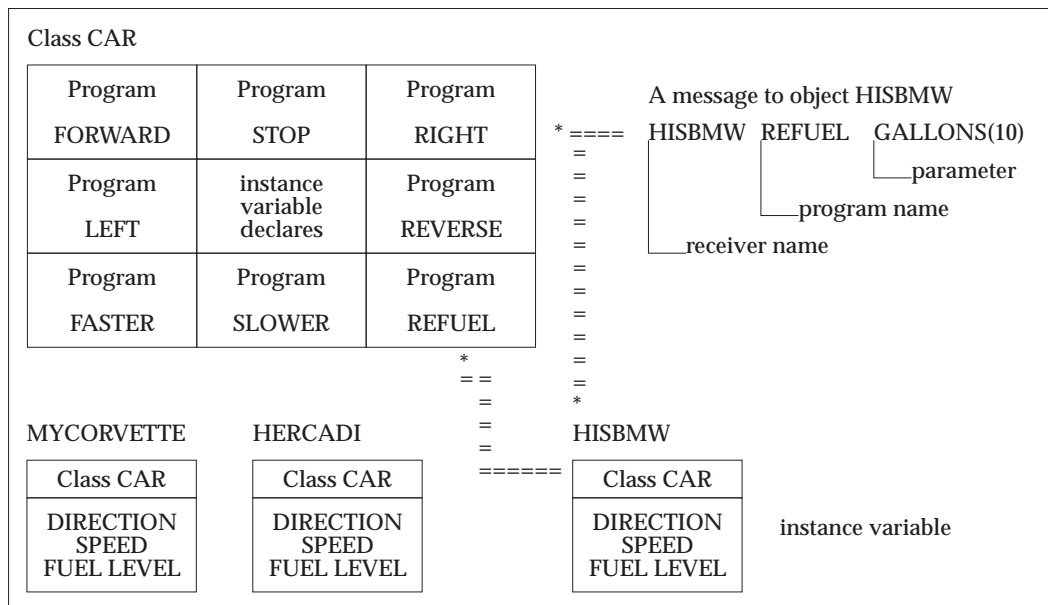


Figure 3-58 Objects as Instances of a Class

A message to car HISBMW to execute program REFUEL is directed to class CAR where program REFUEL is found. The parameter GALLONS(10) is passed to program REFUEL. Each instance of class CAR has its own variables for DIRECTION, SPEED, and FUEL LEVEL, which class CAR declares.

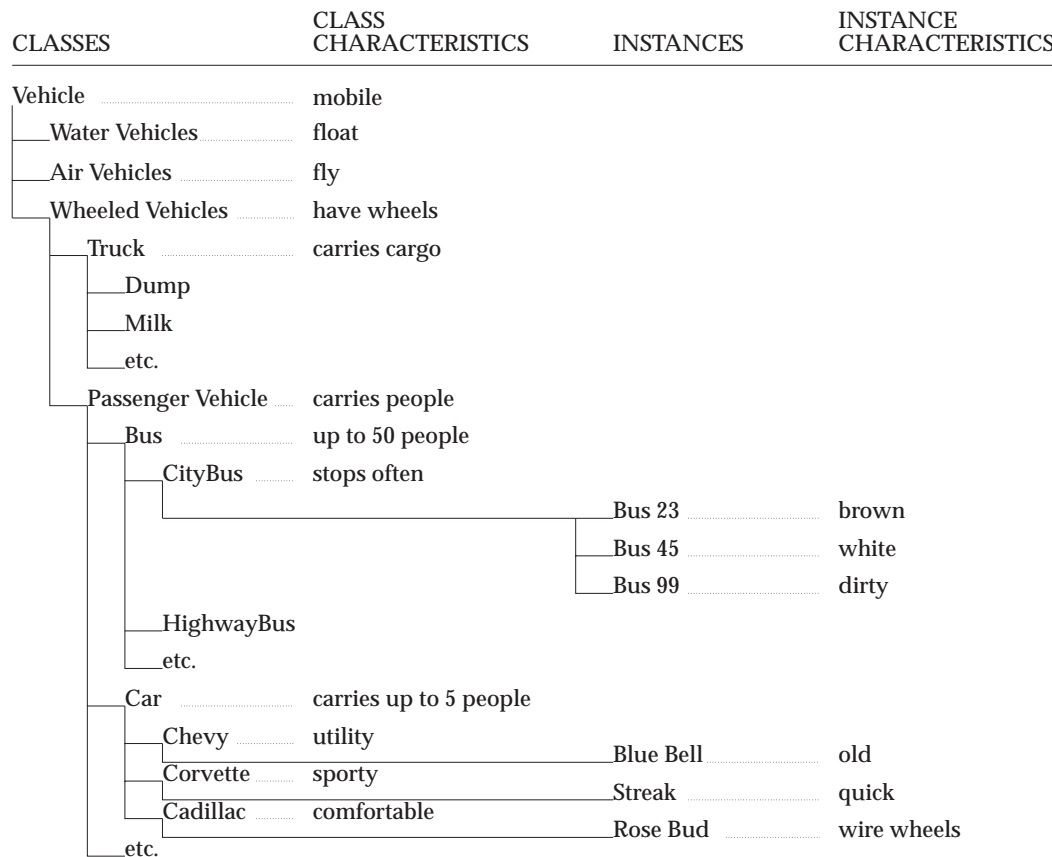
Classes address another problem of interest, creating new objects. In fact, classes can also be thought of as factories that create objects of a particular kind. However, classes cannot always receive messages requesting the creation of instances. Most classes can respond to a message to create new instance objects and to initialize their data values. But for uniformity of concept and processing, classes are themselves objects in most OOPS.

CLASS objects are organized into a hierarchy allowing the definitions of variables and programs to be inherited from more abstract classes to more specific classes. Figure 3-59 (on page 549) illustrates the concept of inheritance. A hierarchical organization provides a context for discussing how things are the same or different and supports the reuse of common programs and variable declarations.

The class of all cars has characteristics unique to cars, but it also inherits characteristics from the categories of Passenger Vehicle and Vehicle. That is, cars have wheels and are mobile, in addition they carry up to five people.

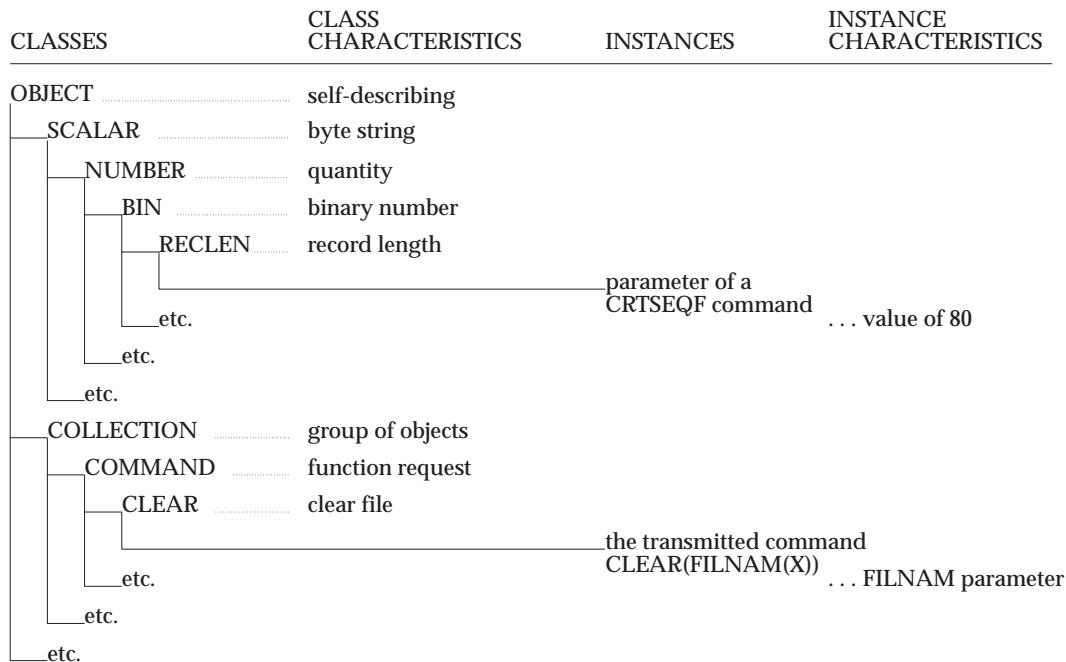
Class Bus is a superclass of class City Bus and a subclass of class Passenger Vehicle. Particular busses, such as Bus 23 and Bus 45, as instances of class City Bus, inherit all of the direct and inherited characteristics of city buses while exhibiting special characteristics of their own; for example, brown and white. Both classes and instances are named for easy reference, such as City Bus, Car, Corvette, and Rose Bud. A car is referred to as a vehicle and a general discussion of vehicles includes all cars.

In DDM architecture, these same techniques are applied to the domain of data as illustrated in Figure 3-60 (on page 550). Classes of data are organized in a hierarchy of inherited structures and behaviors. Instances of these classes are created as needed for particular applications. Both classes and instances are named for easy reference.



**Figure 3-59** Hierarchical Taxonomy

A hierarchical method of describing objects in the application domain brings organization to an apparent chaos of similarities and differences.

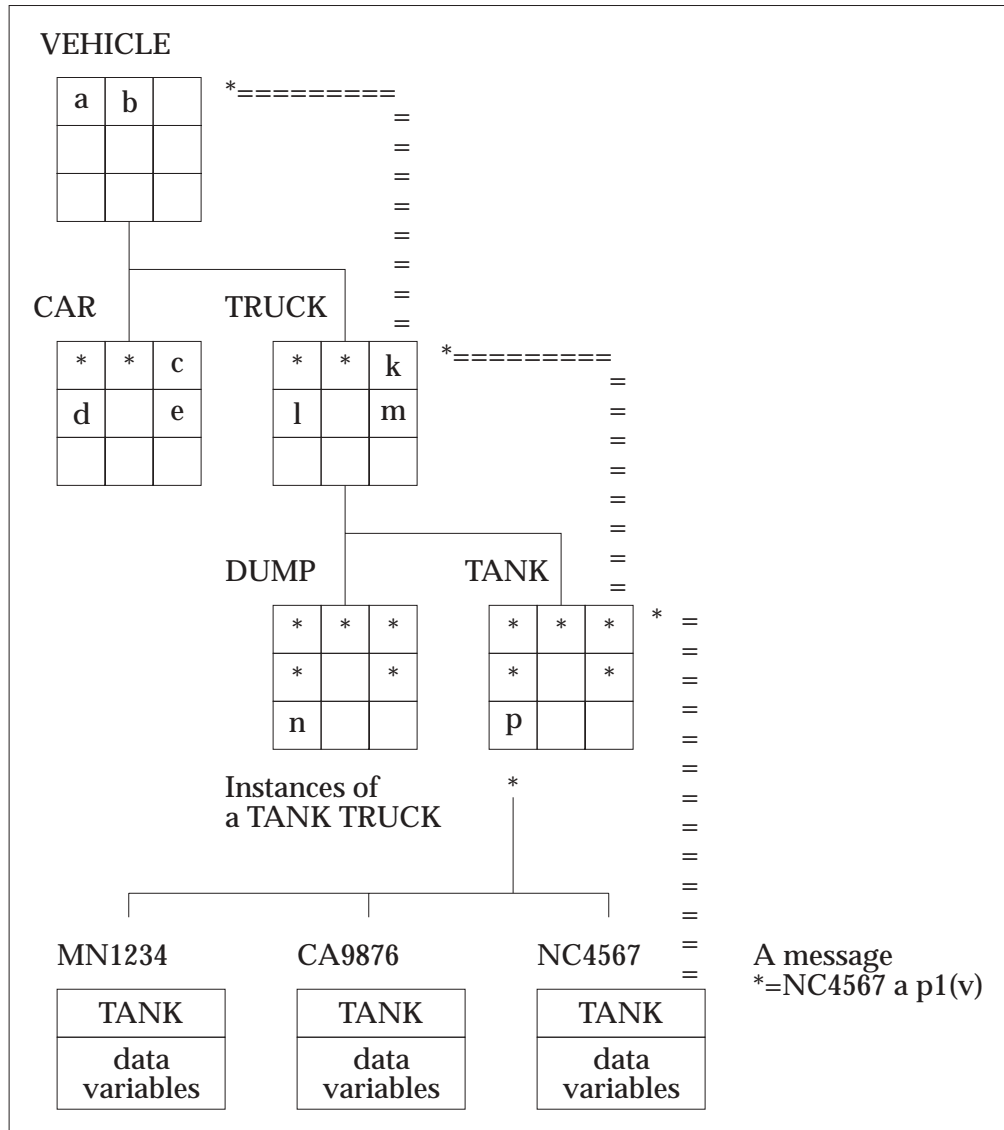


**Figure 3-60** A Sample of the DDM Class Hierarchy

Many classes of objects are similar to other classes of objects, supporting some of the same programs, but perhaps with some additional programs as well. Inheritance is the OOPS mechanism that allows different classes to share programs. If a message requests a program that cannot be found in the receiving object's class, an OOPS with inheritance searches the higher classes in the hierarchy of classes until either the program is found, or there are no higher-level classes, as shown in Figure 3-60.

In actual programming practice, programmers define abstract classes as place holders in the class hierarchy for programs to be shared by more specialized classes. Abstract classes are never instantiated as objects. This process is called generalization. Examples of abstract classes in Figure 3-61 (on page 551) are the classes VEHICLE and TRUCK. The process of defining subclasses with additional capabilities is called specialization.

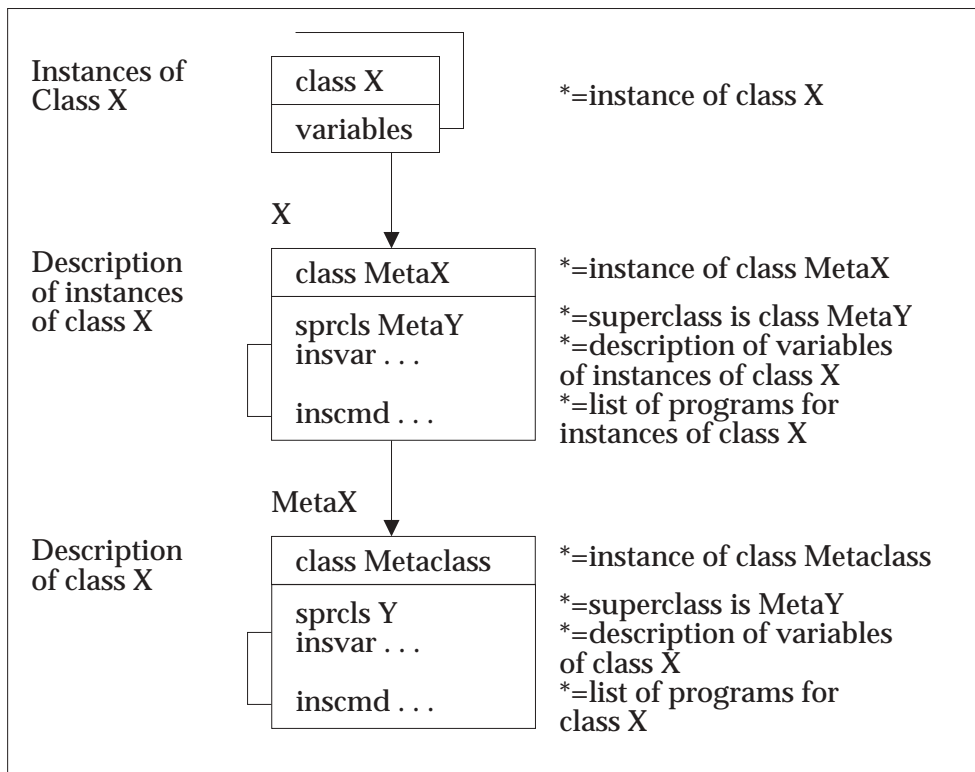
Abstract classes also encourage programmers to write programs that are insensitive to the types of objects being operated on or contained within the data portion of an object. For example, the abstract class SET implements the protocol of mathematical sets without regard for what kind of objects are in the set. This property is called polymorphism, and it serves to eliminate the need to reimplement the protocol of sets for every object that can be aggregated into a set. This also encourages the use of shared programs.



**Figure 3-61** Inheritance of Programs from More Abstract Classes

The message named NC4567 requests a search for program (a) and is sent to TANK TRUCK. It starts at class TANK and proceeds up the superclass hierarchy until the program is found in class VEHICLE.

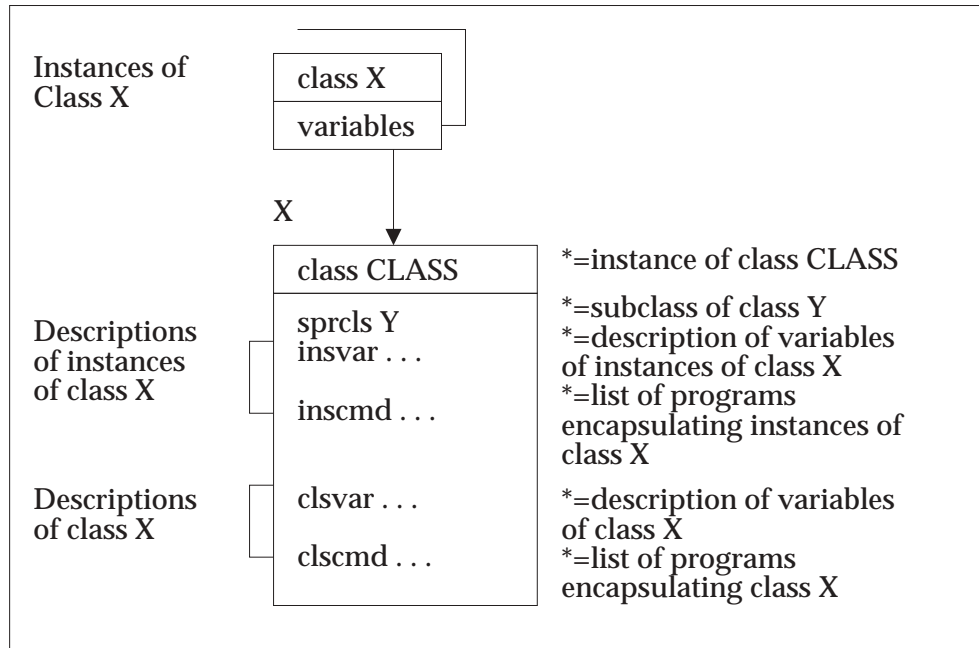
Since classes are themselves objects, each class is an instance of a class called a metaclass. In Smalltalk, an object-oriented programming language, metaclasses are also objects leading to the relationships among instances, classes, and metaclasses illustrated in Figure 3-62 (on page 552).



**Figure 3-62** Smalltalk Instance, Class, and Metaclass Relationships

Class CLASS is a superclass of all classes and metaclasses, allowing them to share in the common definition of classes.

To simplify these relationships in the DDM architecture, classes and metaclasses are combined as a single CLASS object with a single set of variables, as illustrated by Figure 3-63 (on page 553). Thus, the class describes the variables that can exist in a class and specifies the list of programs for the class.



**Figure 3-63** DDM Instance, Class, and Metaclass Relationships

A major benefit of encapsulation is that objects are black boxes whose implementation details are hidden from external view. In particular, the programs developed for each class (or abstract class) of objects can be written in any programming language. While OOPS concepts are useful as a means of specifying program interfaces, like the DDM architecture, DDM products must be carefully integrated with an existing operating system and meet stringent performance criteria. Thus, while the DDM architecture is modeled and documented using OOPS concepts, DDM product designers are free to implement the architecture in ways that best suit each product.

**SEE ALSO**

**Semantic**            *CONCEPTS* (on page 243)  
                           *STRLYR* (on page 890)

**NAME**

OPNQFLRM — Open Query Failure

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2212'  
**Length** \*  
**Class** CLASS  
**Sprcls** RPYMSG - Reply Message

Open Query Failure (OPNQFLRM) Reply Message indicates that the OPNQRY command failed to open the query. The reason that the target relational database was unable to open the query is reported in an SQLCARD reply data object.

Whenever an OPNQFLRM is returned, an SQLCARD object must also be returned following the OPNQFLRM.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'2212'	
<b>rdbnam</b>	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code  8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**cmdrpy** *OPNQRY* (on page 555)  
**Semantic** *FIXROWPRC* (on page 417)  
*LMTBLKPRC* (on page 475)  
*OPNQRY* (on page 555)



**NAME**

OPNQRY — Open Query

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'200C'

**Length** \*

**Class** CLASS

**Sprcls** COMMAND - Command

The Open Query (OPNQRY) command opens a query to a relational database.

**Source System Processing**

The source system determines the location of the RDB:

Local Call the local RDB server.

Remote Send the OPNQRY command to the remote RDB server.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**DSS Carrier: RQSDSS****Target System Processing**

The optional *cmdsrid* parameter uniquely identifies the source of the query.

The optional *dupqryok* parameter specifies whether the target server should allow opening a query for a duplicate cursor. For the definition of a duplicate cursor, refer to Section 4.4.6 of the DRDA Reference.

The *maxblkext* parameter specifies the maximum number of extra blocks of reply data objects and reply messages that the requester is capable of receiving in response to the OPNQRY command.

The *monitor* parameter allows the requester to request a set of events or activities to be monitored on the target server. Refer to *MONITOR* (on page 521) for the list of supported events.

The *pkgnamcsn* parameter specifies the fully qualified package name, the package's consistency token, and a section number within the package containing the query being opened.

The *qryblkctl* parameter controls whether fixed row query protocols must be forced on the opened database cursor. If this parameter is not specified in the query being opened, then the query protocol in use is selected as specified in the package. (More information is in the BGNBND and QRYBLKCTL descriptions.) Regardless of which query protocol the target SQLAM selects for use, the query protocol selected is returned to the source SQLAM as a parameter of the OPNQRYRM.

The *qryblksz* parameter specifies the query block size for the reply data objects and reply messages being returned for this command. The target SQLAM must conform with the specified

query block size. See *QRYBLK* (on page 670) for a definition of a query block.

The *qryclsimp* parameter controls whether the target server implicitly closes a non-scrollable query upon end of data (SQLSTATE 02000) in association with the cursor type.

The *qryclsrls* parameter specifies whether the target server should release all read locks that are held by the cursor when it eventually gets closed either implicitly or explicitly. The target server may not necessarily be able to release all read locks held by a cursor even when requested to do so by the source server since they may be held for other operations or activities.

The *qyrowset* parameter specifies whether a rowset of rows is to be returned with the command. This is only honored for non-dynamic scrollable cursors (*QRYATTNSNS* not equal to *QRYNSSDYN*) and for non-scrollable cursors conforming to the limited block query protocol. The target server fetches no more than the requested number of rows. It may fetch fewer rows if it is restricted by extra query block limits, or if a fetch operation results in a negative SQLSTATE or an SQLSTATE of 02000.

The *qyrowset* parameter is ignored if the cursor is enabled for rowset positioning in support of multi-row fetch.

The *rdbnam* parameter specifies the name of the RDB that the ACCRDB command accesses. If the *rdbnam* parameter is specified, then its value must be the same as the *rdbnam* parameter specified on the ACCRDB command.

The TYPDEFNAM command data object specifies the name of the data type to data representation mapping definitions that the source SQLAM uses when sending command data objects for this command.

The TYPDEFOVR command data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type to data representation mapping definitions for the command data objects for this command.

If the query has input variables, an SQLDTA command data object is sent following the command to describe and carry the input variable data.

The SECTKNOVR command data object is sent by the intermediate server, if the SQLDTA command data object is encrypted. The SECTKNOVR contains the encryption seed and the encryption token used to encrypt the SQLDTA object. The SECTKNOVR DSS is encrypted and must be sent prior to the encrypted SQLDTA command data object.

The TYPDEFNAM reply data object specifies the name of the data type to data representation mapping definitions that the target SQLAM uses when sending the QRYDSC (for this command and the following CNTQRY commands for the query this command opened) or the SQLCARD object.

The TYPDEFOVR reply data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type to data representation mapping definitions for the QRYDSC (for this command and the following CNTQRY commands for the query this command opened) or the SQLCARD object.

Once the query is opened, it remains open until it is either terminated explicitly by a CLSQRY command, or terminated implicitly by:

- A rollback process:
  - A rollback (RDBRLLBCK) command
  - An SQL ROLLBACK statement that is executed by either the EXCSQLSTT or EXCSQLIMM command

- A general backout process
- A commit process in which the SQL cursor did not specify the HOLD clause:
  - A commit (RDBCMM) command
  - An SQL COMMIT statement that is executed by either the EXCSQLSTT or the EXCSQLIMM command
  - A general commit process
- Any reply message (except OPNQRYRM or QRYPOPRM) returned following an OPNQRY or CNTQRY command. If a reply message is returned to terminate the query, the reply message is always followed by an SQLCARD object.

If there are LOBs in the output, the following applies.

The *outovropt* parameter specifies the way in which OUTOVR objects can be flowed with CNTQRY commands.

- If *outovrfrs*, the requester can send an OUTOVR object with the first CNTQRY for the query. After the first CNTQRY, the target will return an error if an OUTOVR object is sent.
- If *outovrany*, the requester can send an OUTOVR object with any CNTQRY for the query.

If no OUTOVR is sent to the target, the target fetches the data using the Description of the data given by the last OUTOVR object sent with a CNTQRY command, or, if none was previously sent, the QRYDSC returned with the OPNQRYRM is used.

Normal completion of the OPNQRY command suspends the query and returns an OPNQRYRM followed by the query description (QRYDSC), followed by zero or more QRYDTA objects, depending on which query protocol the target server selects, the type of blocking to be used for the cursor (QRYBLKTYP), and whether the query involves data types that can only be resolved at the target. For example, if the answer set involves LOB data types, then no QRYDTA objects are returned until the first CNTQRY command is received, along with an OUTOVR object that indicates whether the target is to return such columns as LOB locators or as LOB data values.

The SECTKNOVR reply data object is sent by the intermediate server, if the QRYDTA reply data object is encrypted. The SECTKNOVR contains the encryption seed and the encryption token used to encrypt the QRYDTA object. The SECTKNOVR DSS is encrypted and must be sent prior to the encrypted QRYDTA reply data object.

An RDBUPDRM reply message may be returned in the reply chain in accordance with DRDA Update Control (UP) rules (refer to the DRDA Reference for details).

The OPNQRYRM indicates which query protocol the target SQLAM has selected to use for the query.

Normal completion of this command can return an SQLDARD reply object. To get an SQLDA descriptor area for the executed SQL query, the RTNSQLDA and TYPSQLDA instance variables define if and how the descriptors are returned.

The RTNSQLDA instance variable controls whether to return an SQLDA descriptor area for the output variables for the executed SQL query.

The TYPSQLDA instance variable determines the type of descriptive information to be returned for output variables. Three types of SQLDA descriptor area can be requested: a light, standard, or extended version. Each type provides a different set of descriptive information. The light SQLDA provides minimal descriptive information. The standard SQLDA provides the same descriptive information as was defined for SQLAM level 6 and earlier. The extended SQLDA provides additional descriptive information required by certain types of API such as JDBC.

The SQLDA types are defined by the SQLDARD FD:OCA descriptor. Refer to the DRDA Reference for a description of an SQLDARD.

If the target server ignores the RTNSQLDA parameter, then the target server returns no SQLDARD object to the requester.

### Exceptions

Exception conditions the RDB detects are reported in the SQLCARD or QRYDTA reply data objects.

For non-scrollable cursors, if the last of the answer set data is being returned, and the target server has determined that it can implicitly close the query based on the *qryclsimp* parameter and the cursor type upon SQLSTATE 02000, or some RDB-detected error condition has occurred, and all of the following are true:

- The OPNQRYRM and QRYDSC have been returned.
- LMTBLKPRC query protocols are being used.
- DRDA rules do not prohibit the ENDQRYRM and SQLCARD objects from being returned at this time.

Then an ENDQRYRM and an SQLCARD are returned after any QRYDSC or QRYDTA reply data objects, and the query is terminated. If DRDA rules do not permit the return of the ENDQRYRM and SQLCARD objects, then the objects are saved and the query suspended (see the DRDA Query Data or Result Set Termination rules (QT) and the DRDA Chaining rules (CH) in the DRDA Reference for detailed information).

For scrollable cursors, the query is not terminated when the last of the answer set data is returned. An RDB-detected error condition can terminate the query and result in the return of the ENDQRYRM and SQLCARD. Otherwise, a CLSQRY command from the source SQLAM is required to close the query.

If the target RDB detects an error condition preventing the query from being opened, and the OPNQRYRM has not been returned, then the command is rejected with an OPNQFLRM followed by an SQLCARD object.

If the bind process is active, then the command is rejected with the PKGBPARAM and the query is left in the not open state.

If the query is already open (by a previous OPNQRY command), then the command is rejected with the QRYPOPRM. The query remains suspended.

If the data contained in the *fdodta* parameter of the SQLDTA command data object is not properly described, then the command is rejected with the DTAMCHRM followed by an SQLCARD object, and the query is terminated.

If the target SQLAM is unable to generate a valid FD:OCA descriptor from the descriptor contained in the *fdodsc* parameter of the SQLDTA command data object, then the command is rejected with the DSCINVRM, and the query is terminated.

If access to the specified RDB is not currently obtained, then the command is rejected with the RDBNACRM.

If source system is not a DDM Level 6 system and if the data being returned is of an SQLTYPE not supported by the source system, then the target system rejects the command.

**Source System Reply Processing**

Return any reply messages and data objects to the requester.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**Package Name and Consistency Token**

With SQLAM level 7, the PKGNAMCSN is changed to an optional instance variable. If specified, the values are used as the default package name and consistency token for the connection. If not specified on the command, the PKGSN is required to identify the section number. In this case, the default package name, consistency token and specified section number is used for the request. If PKGNAMCSN was never specified on the connection, a PRCCNVRM is returned.

clsvar	NIL	
insvar	CLASS INSTANCE VARIABLES	
length	*	
class	X'200C'	
cmdsrcid	INSTANCE_OF OPTIONAL MINLVL	CMDSRCID - Command Source Identifier 7
dupqryok	INSTANCE_OF OPTIONAL MINLVL	DUPQRYOK - Duplicate Query Allowed 7
maxblkext	INSTANCE_OF OPTIONAL DFTVAL	MAXBLKEXT - Maximum Number of Extra Blocks 0
monitor	INSTANCE_OF OPTIONAL MINLVL	MONITOR - Monitor Events 7
outovropt	INSTANCE_OF OPTIONAL	OUTOVROPT - Output Override Option
pkgnamcsn	INSTANCE_OF  MINLVL OPTIONAL MTLEXC	PKGNAMCSN - RDB Package Name, Consistency Token, and Section Number 7 Required if PKGSN not specified. PKGSN
pkgsn	INSTANCE_OF MINLVL OPTIONAL MTLEXC	PKGSN - RDB Package Section Number 7 Required if PKGNAMCSN not specified. PKGNAMCSN
qryblkctl	INSTANCE_OF ENUVAL DFTVAL	QRYBLKCTL - Query Block Protocol Control X'2410' - FRCFIXROW - Force Fixed Row Query Protocol ''

	NOTE	Means that the value specified in the package for this parameter is used to determine which query protocol is used for this query.
	OPTIONAL	
qryblksz	INSTANCE_OF REQUIRED	QRYBLKSZ - Query Block Size
qryclsimp	INSTANCE_OF OPTIONAL MINLVL	QRYCLSIMP - Query Close Implicit 7
qryclsrls	INSTANCE_OF OPTIONAL MINLVL	QRYCLSRLS - Query Close Lock Release 7
qryrowset	INSTANCE_OF OPTIONAL IGNORABLE MINLVL	QRYROWSET - Query Rowset Size For non-dynamic scrollable cursors. Otherwise. 7
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
rtnsqlda	INSTANCE_OF OPTIONAL IGNORABLE MINLVL	RTNSQLDA - Return SQL Descriptor Area 7
typsqlda	INSTANCE_OF OPTIONAL IGNORABLE  MINLVL NOTE ENUVAL NOTE ENUVAL NOTE ENUVAL NOTE DFTVAL	TYPESQLDA - Type of SQL Descriptor  If RTNSQLDA is ignored, TYPESQLDA is ignored as well. Otherwise, TYPESQLDA may not be ignored. 7 Supported values: 0 Request standard output SQLDA. 2 Request light output SQLDA. 4 Return extended output SQLDA. 0
clscmd	NIL	
inscmd	NIL	
cmddta		<b>COMMAND OBJECTS</b>
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL MINLVL ENUVAL	TYPDEFNAM - Data Type Definition Name 'QTDSQL370' 'QTDSQL400' 'QTDSQLX86' 'QTDSQLASC' 4 'QTDSQLVAX'

	MINLVL	4
	DFTVAL	''
	NOTE	The default means the value received on ACCRDB command is used.
	OPTIONAL	
X'0035'	INSTANCE_OF	TYPDEFOVR - TYPDEF Overrides
	DFTVAL	''
	NOTE	The default means the value received on ACCRDB command is used.
	OPTIONAL	
X'190B'	INSTANCE_OF	SECTKNOVR - SECTKN Override
	OPTIONAL	
	NOTE	This is sent by the intermediate server, if the SQLDTA command data object is encrypted. This must precede the SQLDTA command data object.
X'2412'	INSTANCE_OF	SQLDTA - SQL Program Variable Data
	OPTIONAL	
	NOTE	Specified when the query has input variables.
<b>rpydta</b>		<b>REPLY OBJECTS</b>
X'002F'	INSTANCE_OF	TYPDEFNAM - Data Type Definition Name
	ENUVAL	'QTDSQL370'
	ENUVAL	'QTDSQL400'
	ENUVAL	'QTDSQLX86'
	ENUVAL	'QTDSQLASC'
	MINLVL	4
	ENUVAL	'QTDSQLVAX'
	MINLVL	4
	DFTVAL	''
	NOTE	The default means the value received on ACCRDBRM is used.
	NOTE	Specifies the TYPDEFNAM used for the QRYDSC reply data object. The value applies to all the data returned in response to a query, including the data returned in response to the CNTQRY commands against the database cursor opened by this command as well as to every EXTDTA block returned for the query.
X'0035'	INSTANCE_OF	TYPDEFOVR - TYPDEF Overrides
	DFTVAL	''
	NOTE	The default means the value received on ACCRDBRM is used.
	NOTE	Specifies the TYPDEFOVR used for the QRYDSC reply data object, the value applies to all the data returned in response to a query, including the data returned in response to the CNTQRY commands against the database cursor opened by this command as well as every EXTDTA block returned for the query.

X'190B'	INSTANCE_OF OPTIONAL NOTE	SECTKNOVR - SECTKN Override  This is sent by the intermediate server, if the QRYDTA reply data object is encrypted. This must precede the QRYDTA reply data object.
X'1C00'	INSTANCE_OF OPTIONAL MINLVL	MONITORRD - Monitor Reply Data  7
X'1C03'	INSTANCE_OF OPTIONAL MINLVL	MGRLVLOVR - Manager Level Overrides  7
X'2408'	INSTANCE_OF OPTIONAL NOTE  REPEATABLE NOTE	SQLCARD - SQL Communications Area Reply Data The SQLCARD object cannot be returned without also returning a reply message. The reply message returned with SQLCARD always precedes the SQLCARD object.  SQLCARD can be sent at most twice. If SQLCARD is sent twice, the first SQLCARD must follow and be associated with the OPNQRYRM.
X'241A'	INSTANCE_OF REQUIRED REPEATABLE NOTE	QRYDSC - Query Answer Set Description  Contains the description, or a portion of the description, of the answer set data.
X'241B'	INSTANCE_OF OPTIONAL REPEATABLE NOTE NOTE	QRYDTA - Query Answer Set Data  Can be returned only if LMTBLKPRC are being used. If a non-terminating error occurs, the QRYDTA object only contains an SQLCARD that reports the error.
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'220D'	INSTANCE_OF	ABNUOWRM - Abnormal End Unit of Work Condition
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported
X'220A'	INSTANCE_OF	DSCINVRM - Invalid Description
X'220E'	INSTANCE_OF	DTAMCHRM - Data Descriptor Mismatch
X'220B'	INSTANCE_OF	ENDQRYRM - End of Query
X'1253'	INSTANCE_OF	OBJNSPRM - Object Not Supported
X'2212'	INSTANCE_OF	OPNQFLRM - Open Query Failure
X'2205'	INSTANCE_OF	OPNQRYRM - Open Query Complete
X'2209'	INSTANCE_OF	PKGBPARM - RDB Package Binding Process Active



X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'220F'	INSTANCE_OF	QRYPOPRM - Query Previously Opened
X'2204'	INSTANCE_OF	RDBNACRM - RDB Not Accessed
X'2218'	INSTANCE_OF	RDBUPDRM - RDB Update Reply Message
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limits Reached
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'125F'	INSTANCE_OF	TRGNSPRM - Target Not Supported
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

**SEE ALSO**

<b>insvar</b>	<p><i>CNTQRY</i> (on page 222)</p> <p><i>MAXBLKEXT</i> (on page 494)</p> <p><i>MONITOR</i> (on page 521)</p> <p><i>OUTOVROPT</i> (on page 576)</p> <p><i>PKGNAMECSN</i> (on page 596)</p> <p><i>PKGSN</i> (on page 606)</p> <p><i>QRYBLKCTL</i> (on page 672)</p> <p><i>QRYBLKSZ</i> (on page 678)</p> <p><i>QRYCLSRLS</i> (on page 682)</p> <p><i>QRYROWSET</i> (on page 697)</p> <p><i>RDBNAM</i> (on page 736)</p>
<b>inscmd</b>	<i>SQLAM</i> (on page 847)
<b>cmddta</b>	<p><i>ACCRDB</i> (on page 42)</p> <p><i>SQLDTA</i> (on page 860)</p> <p><i>TYPDEFNAM</i> (on page 1027)</p> <p><i>TYPDEFOVR</i> (on page 1030)</p>
<b>rpydta</b>	<p><i>ACCRDBRM</i> (on page 48)</p> <p><i>CNTQRY</i> (on page 222)</p> <p><i>EXTDTA</i> (on page 397)</p> <p><i>LMTBLKPRC</i> (on page 475)</p> <p><i>MONITORRD</i> (on page 523)</p> <p><i>OPNQRYRM</i> (on page 566)</p> <p><i>QRYDSC</i> (on page 685)</p> <p><i>QRYDTA</i> (on page 686)</p> <p><i>SQLCARD</i> (on page 855)</p> <p><i>TYPDEFNAM</i> (on page 1027)</p> <p><i>TYPDEFOVR</i> (on page 1030)</p>
<b>cmdrpy</b>	<p><i>ABNUOWRM</i> (on page 39)</p> <p><i>AGNPRMRM</i> (on page 65)</p> <p><i>CMDATHRM</i> (on page 171)</p> <p><i>CMDCHKRM</i> (on page 173)</p> <p><i>CMDNSPRM</i> (on page 176)</p> <p><i>DSCINVRM</i> (on page 298)</p> <p><i>DTAMCHRM</i> (on page 320)</p> <p><i>ENDQRYRM</i> (on page 342)</p> <p><i>OBJNSPRM</i> (on page 542)</p> <p><i>OPNQFLRM</i> (on page 554)</p>

*OPNQRYRM* (on page 566)  
*PKGBPARM* (on page 586)  
*PRCCNVRM* (on page 625)  
*PRMNSPRM* (on page 633)  
*QRYPOPRM* (on page 691)  
*RDBNACRM* (on page 734)  
*RDBUPDRM* (on page 751)  
*RSCLMTRM* (on page 778)  
*SYNTAXRM* (on page 989)  
*TRGNSPRM* (on page 1022)  
*VALNSPRM* (on page 1057)

**Semantic**

*ACCRDB* (on page 42)  
*APPSRCCR* (on page 86)  
*BGNBND* (on page 110)  
*CCSIDMGR* (on page 150)  
*CLASS* (on page 158)  
*CLSQR*Y (on page 165)  
*CMDSRCID* (on page 178)  
*CNTQR*Y (on page 222)  
*COMMAND* (on page 240)  
*DSCINVRM* (on page 298)  
*DTAMCHRM* (on page 320)  
*DUPQR*YOK (on page 322)  
*ENDBND* (on page 336)  
*ENDQR*YRM (on page 342)  
*EXCSQLSTT* (on page 381)  
*EXCSQLIMM* (on page 371)  
*FIXROWPRC* (on page 417)  
*FRCFIXROW* (on page 424)  
*ISOLVLALL* (on page 452)  
*ISOLVLCHG* (on page 453)  
*ISOLVLCS* (on page 454)  
*ISOLVLNC* (on page 455)  
*ISOLVLR*R (on page 456)  
*LMTBLKPRC* (on page 475)  
*MAXBLKEXT* (on page 494)  
*MONITOR* (on page 521)  
*OPNQFLRM* (on page 554)  
*OPNQRY* (on page 555)  
*OPNQRYRM* (on page 566)  
*OUTOVR* (on page 572)  
*OUTOVRANY* (on page 574)  
*OUTOVRFRS* (on page 575)  
*PKGNAMECSN* (on page 596)  
*PKGBPARM* (on page 586)  
*PKGSN* (on page 606)  
*PRCCNVRM* (on page 625)  
*QDDRDBD* (on page 657)  
*QR*YBLK (on page 670)  
*QR*YBLKCTL (on page 672)  
*QR*YBLKFCT (on page 675)  
*QR*YBLKSZ (on page 678)

*QRYBLKTYP* (on page 679)  
*QRYCLSRLS* (on page 682)  
*QRYDSC* (on page 685)  
*QRYDTA* (on page 686)  
*QRYPOPRM* (on page 691)  
*QRYROWSET* (on page 697)  
*RDBCMM* (on page 728)  
*RDBNACRM* (on page 734)  
*RDBRLLBCK* (on page 745)  
*RDBUPDRM* (on page 751)  
*RQSDSS* (on page 774)  
*SECTKNOVR* (on page 822)  
*SQL* (on page 835)  
*SQLAM* (on page 847)  
*SQLDTA* (on page 860)  
*TYPDEFNAM* (on page 1027)  
*TYPDEFOVR* (on page 1030)

**NAME**

OPNQRYSM — Open Query Complete

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2205'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

Open Query Complete (OPNQRYSM) Reply Message indicates to the requester that an OPNQRY or EXCSQLSTT command completed normally, and that a query process has been initiated. It also indicates the type of query protocol and cursor used for the query.

When an EXCSQLSTT contains an SQL statement that invokes a stored procedure, and the procedure completes, an OPNQRYSM is returned for each answer set.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'2205'	
<b>qryattscr</b>	INSTANCE_OF OPTIONAL MINLVL	QRYATTSCR - Query Attribute for Scrollability 7
<b>qryattset</b>	INSTANCE_OF OPTIONAL MINLVL	QRYATTSET - Query Attribute for Rowset 7
<b>qryattsns</b>	INSTANCE_OF OPTIONAL MINLVL	QRYATTSNS - Query Attribute for Sensitivity 7
<b>qryattupd</b>	INSTANCE_OF OPTIONAL MINLVL	QRYATTUPD - Query Attribute for Updatability 7
<b>qryblkfct</b>	INSTANCE_OF OPTIONAL MINLVL	QRYBLKFCT - Query Blocking Factor 7
<b>qryblktyp</b>	INSTANCE_OF OPTIONAL MINLVL	QRYBLKTYP - Query Block Type 7
<b>qryinsid</b>	INSTANCE_OF REQUIRED MINLVL	QRYINSID - Query Instance Identifier 7
<b>qryprctyp</b>	INSTANCE_OF	QRYPRCTYP - Query Protocol Type

	REQUIRED	
sqlcsrhd	INSTANCE_OF OPTIONAL	SQLCSRHLD - Hold Cursor Position
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 0 - INFO - Information Only Severity Code
clscmd	NIL	
inscmd	NIL	

**SEE ALSO**

<b>cmddda</b>	<i>CNTQRY</i> (on page 222)
<b>cmdrpy</b>	<i>EXCSQLSTT</i> (on page 381) <i>OPNQRY</i> (on page 555)
<b>insvar</b>	<i>CNTQRY</i> (on page 222) <i>QRYATTSCR</i> (on page 665) <i>QRYATTSENS</i> (on page 667) <i>QRYATTUPD</i> (on page 669) <i>QRYBLKFCT</i> (on page 675) <i>QRYINSID</i> (on page 688) <i>QRYPRCTYP</i> (on page 692) <i>SQLCSRHLD</i> (on page 858) <i>SVRCOD</i> (on page 911) <i>SRVDGN</i> (on page 873)
<b>rpydta</b>	<i>OPNQRY</i> (on page 555)
<b>Semantic</b>	<i>CLASS</i> (on page 158) <i>CNTQRY</i> (on page 222) <i>DSCSQLSTT</i> (on page 304) <i>EXCSQLSTT</i> (on page 381) <i>FIXROWPRC</i> (on page 417) <i>LMTBLKPRC</i> (on page 475) <i>OPNQRY</i> (on page 555) <i>OPNQRYRM</i> (on page 566) <i>OUTOVR</i> (on page 572) <i>OUTOVRANY</i> (on page 574) <i>OUTOVRFRS</i> (on page 575) <i>QDDRDBD</i> (on page 657) <i>QRYATTSET</i> (on page 666) <i>QRYBLK</i> (on page 670) <i>QRYBLKEXA</i> (on page 674) <i>QRYBLKFCT</i> (on page 675) <i>QRYBLKFLX</i> (on page 676) <i>QRYBLKTYP</i> (on page 679) <i>QRYINSID</i> (on page 688) <i>QRYROWSET</i> (on page 697) <i>RPYMSG</i> (on page 770)

NAME

OPTIONAL — Optional Value Attribute

DESCRIPTION (Semantic)

**Dictionary** QDDPRMD

**Codepoint** X'002D'

**Length** \*

**Class** CLASS

**Sprcls** OBJECT - Self-identifying Data

Optional Value Attribute (OPTIONAL) specifies that support or use of a class, command, parameter, or value is optional according to the rules for subsetting listed in the SUBSETS description.

1. When specified for a command in a class's command list, receivers do not need to support the command for that class. All target servers supporting the class must return the CMDNSPRM if it is not supported.
2. When specified for a parameter in a command's parameter list, the parameter can be sent for that command. All receivers supporting the command must recognize and process the parameter as defined and use the default value if the parameter is not sent.
3. When specified for a parameter in a reply message's parameter list the parameter can be sent for that reply. All receivers must accept the parameter.
4. When specified for a value in a parameter's value list the value need not be sent for the parameter. For example, the hour, minute, and second values of a DATE parameter are optional. All receivers supporting the parameter must recognize the value if it is sent and use the default value if it is not sent.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
<b>length</b>	<b>*</b>
<b>class</b>	<b>X'002D'</b>
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

SEE ALSO

None.

**NAME**

ORDCOL — Ordered Collection

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'004C'

**Length** \*

**Class** CLASS

**Sprcls** COLLECTION - Collection Object

Ordered Collection (ORDCOL) is a collection in which the INSTANCE VARIABLES list orders the collections' elements. Unlike arrays and indexes, no external keys are available to access the elements of an ordered collection.

If an instance variable has an OPTIONAL attribute, it does not need to be specified in the collection, but the variables that are specified must be kept in the required order.

<b>clsvar</b>	NIL
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
<b>length</b>	4
<b>class</b>	X'004C'
<b>clscmd</b>	NIL
<b>inscmd</b>	NIL

**SEE ALSO**

- Semantic**
- ASSOCIATION* (on page 103)
  - FDOOBJ* (on page 411)
  - OUTOVR* (on page 572)
  - PLGINLSE* (on page 612)
  - PLGINLST* (on page 613)
  - SRVLSRV* (on page 876)
  - SRVLST* (on page 878)
  - TYPDEF* (on page 1025)

**NAME**

OSFDCE — OSF DCE Security Mechanism

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

The OSF DCE Security Mechanism (OSFDCE) specifies the use of the Distributed Computing Environment (DCE) security mechanism. This security mechanism is implemented using the *Generic Security Service Application Program Interface (GSS-API)* or an alternative interface that generates the same security context information as the GSS-API.

**SEE ALSO**

**Semantic**        *DCESEC* (on page 252)  
                  *SECMEC* (on page 811)  
                  *SECMGR* (on page 814)  
                  *SECTKN* (on page 820)



**NAME**

OUTEXP — Output Expected

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2111'  
**Length** \*  
**Class** CLASS  
**Sprcls** BOOLEAN - Logical Value

Output Expected indicates whether the requester expects the target SQLAM to return output within an SQLDTARD reply data object as a result of the execution of the referenced SQL statement.

The value of TRUE indicates that the requester expects the command to return output within an SQLDTARD reply data object.

The value of FALSE indicates that the requester does not expect this command to return output within an SQLDTARD reply data object.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	5	
class	X'2111'	
value	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	LENGTH	2
	ENUVAL	X'F1' - TRUE - True State
	NOTE	The requester expects the target SQLAM to return output within an SQLDTARD reply data object.
	ENUVAL	X'F0' - FALSE - False State
	NOTE	The requester does not expect the target SQLAM to return output within an SQLDTARD reply data object.
	DFTVAL	X'F0' - FALSE - False State
	OPTIONAL	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** EXCSQLSTT (on page 381)  
**Semantic** EXCSQLSTT (on page 381)

NAME

OUTOVR — Output Override Descriptor

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'2415'  
**Length** \*  
**Class** CLASS  
**Sprcls** ORDCOL

The OUTOVR object allows a source system to specify the output format for data to be returned as output to an SQL statement or as output from a query. It is specified as command data for an EXCSQLSTT or for a CNTQRY command and causes the target system to return output data in the format contained in the OUTOVR object.

An OUTOVR descriptor contains a valid triplet describing the desired format of each output column whose format is being overridden. If a column's format is not being overridden, a triplet containing a zero LID and a zero length override value (called a default triplet) is sent in place of that column's descriptor.

If no OUTOVR object is specified for an EXCSQLSTT command, the output is returned using the format the data occur in the database.

If no OUTOVR object is specified for a CNTQRY command, the last OUTOVR object specified on a previous CNTQRY command is used, or if no OUTOVR was previously specified, the QRYDSC returned with an OPNQRYRM reply is used.

The OUTOVR object is rejected at the application server in the following cases:

- For a CNTQRY command where the application requester has specified *outovrfrs* and the command is the second or subsequent CNTQRY command for the query.
- For an EXCSQLSTT command where the statement is a stored procedure call.
- For an EXCSQLSTT command where there is no output returned for the command.

<b>clsvar</b>	NIL
<b>insvar</b>	INSTANCE VARIABLES
<b>length</b>	*
<b>class</b>	X'2415'
<b>value</b>	INSTANCE_OF BYTSTRDR - Byte String REQUIRED
<b>clscmd</b>	NIL
<b>inscmd</b>	NIL

SEE ALSO

**cmddta** CNTQRY (on page 222)  
EXCSQLSTT (on page 381)  
**insvar** CNTQRY (on page 222)  
**Semantic** CNTQRY (on page 222)  
EXCSQLSTT (on page 381)  
OPNQRY (on page 555)

*OUTOVRANY* (on page 574)  
*OUTOVRFRS* (on page 575)  
*OUTOVROPT* (on page 576)

**NAME**

OUTOVRANY — Output Override Allowed on Any CNTQRY

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Length** \*

**Class** CONSTANT

On an EXCSQLSTT or OPNQRY command, OUTOVRANY indicates that the query being opened allows an OUTOVR on any CNTQRY command sent by the source system. If the application requester specifies this option, the application server always selects FIXROWPRC for the query.

If an OUTOVR object is sent as command data with a CNTQRY command, the answer set row will be fetched according to the descriptors in the OUTOVR.

If an OUTOVR object is not sent with a CNTQRY command, the answer set row will be fetched according to the descriptors in the last OUTOVR received with a CNTQRY. If no previous CNTQRY command had an OUTOVR command data object, then the answer set row will be fetched according to the descriptors in the QRYDSC returned by the target with the OPNQRYRM.

---

value	2
-------	---

**SEE ALSO**

**Semantic** *OUTOVROPT* (on page 576)

**NAME**

OUTOVRFRS — Output Override Allowed on First CNTQRY

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Length** \*

**Class** CONSTANT

On an OPNQRY or EXCSQLSTT command, OUTOVRFRS indicates that the query being opened allows an OUTOVR on the first CNTQRY command sent by the source system.

If an OUTOVR object is sent as command data with the first CNTQRY command, the answer set row and every subsequent row will be fetched according to the descriptors in the OUTOVR.

If an OUTOVR object is not sent with the first CNTQRY command, the answer set row will be fetched according to the descriptors in the QRYDSC returned by the target with the OPNQRYRM.

---

value	1
-------	---

**SEE ALSO**

**Semantic** *OUTOVROPT* (on page 576)

**NAME**

OUTOVRPT — Output Override Option

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2147'  
**Length** \*  
**Class** CLASS  
**Sprcls** STRING - String

OUTOVRPT specifies how OUTOVR objects can be specified with CNTQRY commands.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>INSTANCE VARIABLES</b>	
length	*	
class	X'2147'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	8
	ENUVAL	1 OUTOVRFRS
	NOTE	Output Override Allowed on First CNTQRY.
	ENUVAL	2 OUTOVRANY
	NOTE	Output Override Allowed on Any CNTQRY.
	DFTVAL	1 OUTOVRFRS
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** EXCSQLSTT (on page 381)  
 OPNQRY (on page 555)  
**Semantic** CNTQRY (on page 222)

**NAME**

OWNER — Package Owner Authorization Identifier

**DESCRIPTION (Semantic)****Dictionary** QDDPRMD**Length** \***Class** CONSTANT

Package owner authorization identifier (OWNER) value indicates that the authorization identifier used for the execution of a dynamic SQL package is the authorization identifier of the person who owns the package.

---

value	1
-------	---

**SEE ALSO****insvar** *PKGATHRUL* (on page 581)**Semantic** *PKGATHRUL* (on page 581)

**NAME**

PASSWORD — Password

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD  
**Codepoint** X'11A1'  
**Length** \*  
**Class** CLASS  
**Sprcls** NAME - Name

Password at the Target System (PASSWORD) specifies the password associated with the defined end-user name.

With certain SECMECs, the password can be encrypted.

The target server can optionally *fold* the password if it is not in the correct form for the target security manager. Folding is the process of modifying the characters from one form to another. For instance, from mixed case into all upper or all lowercase.

DDM architecture treats the PASSWORD as a byte string. The SECMGR is aware of and has knowledge of the contents of the PASSWORD. DDM does not know or care about the specific contents of this field. If the reader needs to know the actual contents, see *USRSECOVR* (on page 1050).

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'11A1'	
value	INSTANCE_OF	CHRSTRDR - Character String Data Representation
	MINLEN	1
	MAXLEN	255
	REQUIRED	
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *SECCHK* (on page 800)  
**Semantic** *SECCHK* (on page 800)  
*SYNCMNI* (on page 931)  
*USRSECOVR* (on page 1050)



**NAME**

PKGATHKP — Package Authorizations Keep

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2425'

**Length** \*

**Class** codpnt

Package Authorizations Keep (PKGATHKP) specifies that the target relational database must keep the existing package authorizations when replacing a package.

If the new package has a new owner either because the PKGOWNID parameter was specified or because the BGNBND requester is a different end user than the previous package owner then:

1. The previous package owner retains EXECUTE privilege on the new package but loses the implicit privileges of ownership.
2. All users who were given EXECUTE privilege by the previous package owner retain the privilege, but the privileges are changed to indicate that they were given by the new package owner.
3. The new package owner is automatically given the EXECUTE privilege on the new package and the implicit privileges of ownership.

If the owner of the package is not changed, then all of the existing EXECUTE privileges on the existing package are retained unchanged by the new package.

**SEE ALSO**

**insvar** *PKGATHOPT* (on page 580)

**Semantic** *PKGOWNID* (on page 600)  
*PKGRPLALW* (on page 602)

NAME

PKGATHOPT — Package Authorization Option

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD

**Codepoint** X'211E'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Package Authorization Option (PKGATHOPT) String specifies whether the existing package authorizations are maintained or revoked when a package is being replaced. This parameter only has meaning when PKGRPLOPT(PKGRPLALW) is specified on the BGNBND command and a package currently exists with the same package and version name as that specified by the PKGNAMCT and VRSNAM parameters.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	6	
<b>class</b>	X'211E'	
<b>value</b>	INSTANCE_OF	CODPNTDR - Codepoint Data Representation
	ENUVAL	X'2425' - PKGATHKP - Package Authorizations Keep
	ENUVAL	X'2424' - PKGATHRVK - Package Authorizations Revoked
	DFTVAL	X'2425' - PKGATHKP - Package Authorizations Keep
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

SEE ALSO

**insvar** BGNBND (on page 110)

**Semantic** ENDBND (on page 336)  
 PKGOWNID (on page 600)  
 PKGRPLALW (on page 602)  
 REBIND (on page 753)

**NAME**

PKGATHRUL — Package Authorization Rules

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'213F'

**Length** \*

**Class** CLASS

**Sprcls** BIN - Binary Integer Number

The Package Authorization Rules (PKGATHRUL) Binary Integer Number specifies which authorization identifier to use when dynamic SQL in a package is executed. The possible alternatives are as follows:

- The requester (the person executing the package)
- The owner (the person who owns the package being executed)
- The invoker (for functions or stored procedures, the person whose authorization identifier is associated with the statement or view invoking the function or stored procedure)
- The definer (for functions or stored procedures, the person who created the function or stored procedure)

The value of REQUESTER indicates that the authorization identifier used for the execution of a dynamic SQL package is the package requester's authorization identifier, which is the default value.

The value of OWNER indicates that the authorization identifier used for the execution of a dynamic SQL package is the package owner's authorization identifier, which is the same SQL authorization identifier that is used for static SQL. Thus, this bind option causes static and dynamic SQL at the target to have the same behavior from an authorization point of view.

The value of INVOKER\_REVERT\_TO\_REQUESTER and INVOKER\_REVERT\_TO\_OWNER indicates that the authorization identifier depends on the invocation of the function or stored procedure:

- If the function or stored procedure in a dynamic SQL package is invoked by an SQL statement, the authorization identifier used is that of the invoking SQL statement. For static statements, the authorization identifier is the owner of the package containing the statement. For dynamic statements, the authorization identifier depends on the PKGATHRUL option for the package containing the statement.
- If the function is invoked by a view, the authorization identifier is that of the person who created the view.

If INVOKER\_REVERT\_TO\_REQUESTER is specified and the package is executed but not as a function or stored procedure, the value reverts to REQUESTER.

If INVOKER\_REVERT\_TO\_OWNER is specified and the package is executed but not as a function or stored procedure, the value reverts to OWNER.

The value of DEFINER\_REVERT\_TO\_REQUESTER and DEFINER\_REVERT\_TO\_OWNER indicates that the authorization identifier used for the execution of a dynamic SQL statement in a function or stored procedure is the authorization identifier of the person who created or defined the function or stored procedure.

If DEFINER\_REVERT\_TO\_REQUESTER is specified and the package is executed but not as a function or stored procedure, the value reverts to REQUESTER.

If DEFINER\_REVERT\_TO\_OWNER is specified and the package is executed but not as a function or stored procedure, the value reverts to OWNER.

clsvar	NIL	
insvar	INSTANCE VARIABLES	
length	5	
class	X'213F'	
value	INSTANCE_OF	BINDR - Binary Number Field QDDPRMD
	LENGTH	8
	ENUVAL	0 - REQUESTER - Package Requester Authorization Identifier
	NOTE	The dynamic SQL package's authorization identifier is the package execution requester's authorization identifier.
	ENUVAL	1 - OWNER - Package Owner Authorization Identifier
	NOTE	The dynamic SQL package's authorization identifier is the package owner's authorization identifier.
	ENUVAL	2 - INVOKER_REVERT_TO_REQUESTER - Invoking View/Statement Authorization Identifier
	NOTE	The dynamic SQL package's authorization identifier is the authorization identifier of the statement invoking the function or stored procedure, or is the creator of the view invoking function.
	NOTE	If the package is not invoked as a function or stored procedure, the value reverts to REQUESTER.
	ENUVAL	3 - INVOKER_REVERT_TO_OWNER - Invoking View/Statement Authorization Identifier.
	NOTE	The dynamic SQL package's authorization identifier is the authorization identifier of the statement invoking the function or stored procedure, or is the creator of the view invoking function
	NOTE	If the package is not invoked as a function or stored procedure, the value reverts to OWNER.
	ENUVAL	4 - DEFINER_REVERT_TO_REQUESTER - Authorization Identifier of Function or Stored Procedure Creator
	NOTE	The dynamic SQL package's authorization identifier is the creator of the function or stored procedure.
	NOTE	If the package is not invoked as a function or stored procedure, the value reverts to REQUESTER.
	ENUVAL	5 - DEFINER_REVERT_TO_OWNER - Authorization

	NOTE	Identifier of Function or Stored Procedure Creator The dynamic SQL package's authorization identifier is the creator of the function or stored procedure
	NOTE	If the package is not invoked as a function or stored procedure, the value reverts to OWNER.
	DFTVAL	0
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar**      *BGNBND* (on page 110)  
                  *REBIND* (on page 753)

**NAME**

PKGATHRVK — Package Authorizations Revoked

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2424'

**Length** \*

**Class** codpnt

Package Authorizations Revoked (PKGATHRVK) specifies that the target relational database must revoke the existing package authorizations when replacing an existing package.

If the new package has a new owner either because the PKGOWNID parameter was specified or because the BGNBND requester is a different end user than the previous package owner, then:

1. The previous package owner has all previously held privileges revoked for the new package.
2. All users who were given the EXECUTE privilege by the previous package owner have that privilege revoked for the new package.
3. The new package owner is automatically given the EXECUTE privilege and the implicit ownership privileges on the new package.

If the owner of the package is not changed, then all of the existing EXECUTE privileges on the existing package are revoked except that the package owner retains the EXECUTE privilege.

**SEE ALSO**

**insvar** *PKGATHOPT* (on page 580)

**Semantic** *PKGOWNID* (on page 600)  
*PKGRPLALW* (on page 602)

**NAME**

PKGBNARM — RDB Package Binding Not Active

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2206'  
**Length** \*  
**Class** CLASS  
**Sprcls** RPYMSG - Reply Message

RDB Package Binding Not Active (PKGBNARM) Reply Message indicates that a BNDSQLSTT or ENDBND command was issued when the package binding process was not active for the specified package name. A BGNBND command must precede a BNDSQLSTT or ENDBND command to initiate the package binding process.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'2206'	
<b>rdbnam</b>	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**cmdrpy** *BNDSQLSTT* (on page 136)  
*ENDBND* (on page 336)  
**Semantic** *BNDSQLSTT* (on page 136)  
*ENDBND* (on page 336)

NAME

PKGBPARAM — RDB Package Binding Process Active

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'2209'  
**Length** \*  
**Class** CLASS  
**Sprcls** RPYMSG - Reply Message

RDB Package Binding Process Active (PKGBPARAM) Reply Message indicates that the command cannot be issued when the relational database package binding process is active. The active package binding process must be terminated (see *BGNBND* (on page 110) and *ENDBND* (on page 336)) before the command can be issued.

**DSS Carrier:** RPYDSS

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'2209'	
<b>rdbnam</b>	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrnod</b>	INSTANCE_OF REQUIRED	SVRCOD - Severity Code
	ENUVAL	8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

SEE ALSO

**cmdrpy** *BGNBND* (on page 110)  
*CLSQR*Y (on page 165)  
*CNTQR*Y (on page 222)  
*DRPPK*G (on page 293)  
*DSCRDBTBL* (on page 300)  
*DSCSQLSTT* (on page 304)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSTT* (on page 381)  
*OPNQRY* (on page 555)  
*PRPSQLSTT* (on page 636)  
*REBIND* (on page 753)

**Semantic** *BGNBND* (on page 110)  
*CLSQR*Y (on page 165)  
*CNTQR*Y (on page 222)  
*DRPPK*G (on page 293)  
*DSCRDBTBL* (on page 300)



*DSCSQLSTT* (on page 304)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSET* (on page 377)  
*EXCSQLSTT* (on page 381)  
*OPNQRY* (on page 555)  
*PRPSQLSTT* (on page 636)  
*REBIND* (on page 753)

**NAME**

PKGCNSTKN — RDB Package Consistency Token

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'210D'

**Length** \*

**Class** CLASS

**Sprcls** FIELD - A Discrete Unit of Data

RDB Package Consistency Token (PKGCNSTKN) verifies that the requester's application and the relational database package are synchronized.

If more than one package has the same fully qualified package name, they cannot have the same consistency tokens.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>name</b>	INSTANCE_OF	BYTSTRDR - Byte String
	LENGTH	8
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	
<b>vldattls</b>	NIL	

**SEE ALSO**

**insvar** *PKGNAMECSN* (on page 596)  
*PKGNAMECT* (on page 598)

**Semantic** *PKGNAME* (on page 594)  
*REBIND* (on page 753)  
*SQL* (on page 835)

**NAME**

PKGDFTC — Package Default CCSIDs for a Column

**DESCRIPTION (Semantic)****Dictionary** QDDBASD**Codepoint** X'119A'**Length** \***Class** CLASS**Sprcls** COLLECTION - Collection Object

Package Default CCSIDs for a Column (PKGDFTC) Collection Object specifies the default CCSIDs used if a character or graphic column is defined by an SQL CREATE or ALTER table statement without having an explicit CCSID specified for the column.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'119A'	
ccsiddbc	INSTANCE_OF OPTIONAL DFTVAL NOTE	CCSIDDBC - CCSID for Double-byte Characters " The default double-byte CCSID the target system defines is used.
ccsidmbc	INSTANCE_OF OPTIONAL DFTVAL NOTE	CCSIDMBC - CCSID for Mixed-byte Characters " The default mixed-byte CCSID defines the target system is used.
ccsidsbc	INSTANCE_OF OPTIONAL DFTVAL NOTE	CCSIDSBC - CCSID for Single-byte Characters " The default single-byte CCSID the target system defines is used.
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO****insvar** *BGNBND* (on page 110)**Semantic** *ENDBND* (on page 336)  
*REBIND* (on page 753)

**NAME**

PKGDFTCST — Package Default Character Subtype

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2125'  
**Length** \*  
**Class** CLASS  
**Sprcls** STRING - String

Package Default Character Subtype (PKGDFTCST) String e specifies the default SQL character subtype used if a character column is defined by an SQL CREATE or ALTER table statement without an explicit subtype being specified.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	6	
class	X'2125'	
value	INSTANCE_OF	CODPNTDR - Codepoint Data Representation
	ENUVAL	X'2432' - CSTSYSDFD - Character Subtype System Default
	ENUVAL	X'2433' - CSTBITS - Character Subtype Bits
	ENUVAL	X'2434' - CSTSBCS - Character Subtype SBCS
	ENUVAL	X'2435' - CSTMBCS - Character Subtype MBCS
	DFTVAL	X'2432' - CSTSYSDFD - Character Subtype System Default
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** ACCRDBRM (on page 48)  
 BGNBND (on page 110)  
**Semantic** ENDBND (on page 336)  
 REBIND (on page 753)

**NAME**

PKGID — RDB Package Identifier

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2109'  
**Length** \*  
**Class** CLASS  
**Sprcls** FIELD - A Discrete Unit of Data

RDB Package Identifier (PKGID) field specifies the identifier of a relational database package. The combination of the RDBCOLID, PKGID, and VRSNAM uniquely identifies a package within the application server site RDB.

Prior to DDM Level 7, the PKGID had a fixed length of 18 characters, including right blank padding if the package identifier is less than 18 characters long. As of DDM Level 7, the PKGID can accommodate a package identifier of up to 255 characters in length, and its format will vary depending on the length of the package identifier:

- Length of package identifier <= 18 characters  
 There is no change to the format of the PKGID. The length of the PKGID remains fixed at 18 which includes any right blank padding if necessary.
- Length of package identifier > 18 characters  
 The length of the PKGID is identical to the length of the package identifier. No right blank padding is required.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
<b>name</b>	INSTANCE_OF	CHRSTRDR - Character String Data Representation
	MINLEN	18
	MAXLEN	255
	MINLVL	8
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>vldattls</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** CHRSTRDR (on page 156)  
 PKGNAM (on page 594)  
 PKGNAMCSN (on page 596)  
 PKGNAMCT (on page 598)  
**Semantic** SQL (on page 835)

NAME

PKGISOLVL — Package Isolation Level

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'2124'  
**Length** \*  
**Class** CLASS  
**Sprcls** STRING - String

Package Isolation Level (PKGISOLVL) string specifies the isolation level used when SQL statements in the package are executed unless some target relational database runtime mechanism overrides them. This parameter does not affect the isolation level used during the package bind process.

When the package is created, the target RDB is allowed to promote the specified isolation level to a level that provides more protection. In this respect, the package isolation levels are listed below with the isolation level that provides the most protection listed first and the isolation level that provides the least protection listed last.

```
ISOLVLR   <-- Most protection
ISOLVLALL
ISOLVLCS
ISOLVLCHG
ISOLVLNC  <-- Least protection
```

The target RDB can promote the specified isolation level to the next higher isolation level in the preceding list that the target RDB supports.

The target RDB cannot demote the specified package isolation level. This would result in exposing the requester to data integrity problems.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	6	
class	X'2124'	
value	INSTANCE_OF	CODPNTDR - Codepoint Data Representation
	ENUVAL	X'2444' - ISOLVLR - Isolation Level Repeatable Read
	ENUVAL	X'2443' - ISOLVLALL - Isolation Level All
	ENUVAL	X'2442' - ISOLVLCS - Isolation Level Cursor Stability
	ENUVAL	X'2441' - ISOLVLCHG - Isolation Level Change
	ENUVAL	X'2445' - ISOLVLNC - Isolation Level No Commit
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar**

*BGNBND* (on page 110)

*REBIND* (on page 753)

NAME

PKGNAM — RDB Package Name

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'210A'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

RDB Package Name (PKGNAM) Scalar Object specifies the fully qualified name of a relational database package. More than one package can have the same fully qualified name. Two packages with the same fully qualified name are differentiated by their version names (VRSNAM) or their consistency tokens (PKGCNSTKN).

The PKGNAM can have one of the following two formats depending on the length of the RDBNAM, RDBCOLID, and PKGID contained therein:

- RDBNAM, RDBCOLID, and PKGID all have a length of 18.  
 This format of the PKGNAM is identical to the sole format used prior to DDM Level 7 where the length is fixed at 58. The use of the SCLDTALEN is disallowed with this format.
- At least one of RDBNAM, RDBCOLID, and PKGID has a length > 18.  
 This format of the PKGNAM mandates the SCLDTALEN to precede each of the RDBNAM, RDBCOLID, and PKGID. With this format, the PKGNAM has a minimum length of 65 and a maximum length of 775.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
<b>length</b>	*	
	NOTE	If the RDBNAM, RDBCOLID, and PKGID all have a length of 18, the SCLDTALEN is disallowed and the length of the PKGNAM will be 58. Otherwise, the SCLDTALEN is mandatory for indicating the length of each of the RDBNAM, RDBCOLID, and PKGID, and the length of the PKGNAM will be between 65 and 775 inclusive.
<b>class</b>	X'210A'	
<b>pkgid</b>	INSTANCE_OF	PKGID - RDB Package Identifier
<b>rdbcolid</b>	INSTANCE_OF	RDBCOLID - RDB Collection Identifier
<b>rdbnam</b>	INSTANCE_OF LENGTH NOTE	CHRSTRDR - Character String Data Representation 18 This is the application server site RDBNAM of the package. The syntax of this field is not validated.
<b>scldtalen</b>	INSTANCE_OF OPTIONAL	SCLDTALEN - Scalar Data Length



NOTE If the length of any one of RDBNAM, RDBCOLID or PKGID exceeds 18, the SCLDTALEN is mandatory and must precede the RDBNAM. Otherwise, the SCLDTALEN is disallowed.

	MINLVL	7
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *DRPPKG* (on page 293)  
*REBIND* (on page 753)

**rdbnam** *CHRSTRDR* (on page 156)

**Semantic** *REBIND* (on page 753)  
*SQL* (on page 835)

NAME

PKGNAMCSN — RDB Package Name, Consistency Token, and Section Number

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'2113'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

RDB Package Name, Consistency Token, and Section Number (PKGNAMCSN) Scalar Object specifies the fully qualified name of a relational database package, its consistency token, and a specific section within the package.

The PKGNAMCSN can have one of the following two formats depending on the length of the RDBNAM, RDBCOLID, and PKGID contained therein:

- RDBNAM, RDBCOLID, and PKGID all have a length of 18.  
 This format of the PKGNAMCSN is identical to the sole format used prior to DDM Level 7 where the length is fixed at 68. The use of the SCLDTALEN is disallowed with this format.
- At least one of RDBNAM, RDBCOLID, and PKGID has a length > 18.  
 This format of the PKGNAMCSN mandates the SCLDTALEN to precede each of the RDBNAM, RDBCOLID, and PKGID. With this format, the PKGNAMCSN has a minimum length of 75 and a maximum length of 785.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
	NOTE	If the RDBNAM, RDBCOLID, and PKGID all have a length of 18, the SCLDTALEN is disallowed and the length of the PKGNAMCSN will be 68. Otherwise, the SCLDTALEN is mandatory for indicating the length of each of the RDBNAM, RDBCOLID, and PKGID, and the length of the PKGNAMCSN will be between 75 and 785 inclusive.
<b>class</b>	X'2113'	
<b>pkgcnstkn</b>	INSTANCE_OF	PKGCNSTKN - RDB Package Consistency Token
<b>pkgid</b>	INSTANCE_OF	PKGID - RDB Package Identifier
<b>pkgsn</b>	INSTANCE_OF	PKGSN - RDB Package Section Number
<b>rdbcolid</b>	INSTANCE_OF	RDBCOLID - RDB Collection Identifier
<b>rdbnam</b>	INSTANCE_OF LENGTH NOTE	CHRSTRDR - Character String Data Representation 18 This is the application server site RDBNAM of the package. The syntax of this field is not validated.

<b>scldtalen</b>	INSTANCE_OF OPTIONAL NOTE	SCLDTALEN - Scalar Data Length  If the length of any one of RDBNAM, RDBCOLID or PKGID exceeds 18, the SCLDTALEN is mandatory and must precede the RDBNAM. Otherwise, the SCLDTALEN is disallowed.
	MINLVL	7
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

<b>insvar</b>	<i>BNDSQLSTT</i> (on page 136) <i>CLSQR</i> Y (on page 165) <i>CNTQR</i> Y (on page 222) <i>DSCSQLSTT</i> (on page 304) <i>EXCSQLIMM</i> (on page 371) <i>EXCSQLSTT</i> (on page 381) <i>OPNQR</i> Y (on page 555) <i>PKGSNLST</i> (on page 607) <i>PRPSQLSTT</i> (on page 636) <i>QRYNOPRM</i> (on page 690) <i>QRYPOPRM</i> (on page 691)
<b>rdbnam</b>	<i>CHRSTRDR</i> (on page 156)
<b>Semantic</b>	<i>CNTQR</i> Y (on page 222) <i>ENDBND</i> (on page 336) <i>LMTBLKPRC</i> (on page 475) <i>OPNQR</i> Y (on page 555) <i>RDBCOLID</i> (on page 732) <i>SQL</i> (on page 835)

NAME

PKGNAMCT — RDB Package Name and Consistency Token

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'2112'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

RDB Package Name and Consistency Token (PKGNAMCT) Scalar Object specifies the fully qualified name of relational database package and its consistency token.

The PKGNAMCT can have one of the following two formats depending on the length of the RDBNAM, RDBCOLID, and PKGID contained therein:

- RDBNAM, RDBCOLID, and PKGID all have a length of 18.  
 This format of the PKGNAMCT is identical to the sole format used prior to DDM Level 7 where the length is fixed at 66. The use of the SCLDTALEN is disallowed with this format.
- At least one of RDBNAM, RDBCOLID, and PKGID has a length > 18.  
 This format of the PKGNAMCT mandates the SCLDTALEN to precede each of the RDBNAM, RDBCOLID, and PKGID. With this format, the PKGNAMCT has a minimum length of 73 and a maximum length of 783.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
	NOTE	If the RDBNAM, RDBCOLID, and PKGID all have a length of 18, the SCLDTALEN is disallowed and the length of the PKGNAMCT will be 66. Otherwise, the SCLDTALEN is mandatory for indicating the length of each of the RDBNAM, RDBCOLID, and PKGID, and the length of the PKGNAMCT will be between 73 and 783 inclusive.
<b>class</b>	X'2112'	
<b>pkgcnstkn</b>	INSTANCE_OF	PKGICNSTKN - RDB Package Consistency Token
<b>pkgid</b>	INSTANCE_OF	PKGID - RDB Package Identifier
<b>rdbcolid</b>	INSTANCE_OF	RDBCOLID - RDB Collection Identifier
<b>rdbnam</b>	INSTANCE_OF LENGTH NOTE	CHRSTRDR - Character String Data Representation 18 This is the application server site RDBNAM of the package. The syntax of this field is not validated.
<b>scldtalen</b>	INSTANCE_OF OPTIONAL	SCLDTALEN - Scalar Data Length

**NOTE** If the length of any one of RDBNAM, RDBCOLID, or PKGID exceeds 18, the SCLDTALEN is mandatory and must precede the RDBNAM. Otherwise, the SCLDTALEN is disallowed.

	MINLVL	7
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *BGNBND* (on page 110)  
*BGNBNDRM* (on page 117)  
*ENDBND* (on page 336)  
*EXCSQLSET* (on page 377)

**rdbnam** *CHRSTRDR* (on page 156)

**Semantic** *ENDBND* (on page 336)  
*PKGATHOPT* (on page 580)  
*PKGRPLALW* (on page 602)  
*PKGRPLOPT* (on page 604)  
*SQL* (on page 835)

NAME

PKGOWNID — Package Owner Identifier

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'2131'  
**Length** \*  
**Class** CLASS  
**Sprcls** NAME - Name

Package Owner Identifier (PKGOWNID) specifies the end-user name (identifier) of the user that is the owner of the package. The target SQLAM is responsible to any authentication and/or verification of the end-user name which the DDM architecture does not define.

The owner of the package is the end-user name (identifier) the target RDB uses for validation of authority to perform the functions represented by the SQL statements being bound or rebound into the package.

It is valid for the PKGOWNID to specify an end-user name (identifier) that is different than the one currently associated with an existing package (however, the package collection identifier cannot be changed).

The new end-user name is assigned as the owner of the replaced package:

- If the PKGRPLOPT(PKGRPLALW) is specified on BGNBND and the bind process results in the package being replaced (see *BNDCRTCTL* (on page 127) and *ENDBND* (on page 336)).
- If the rebind process results in the package being replaced.

See *PKGATHKP* (on page 579) and *PKGATHRVK* (on page 584) (values of the PKGATHOPT parameter) for a description of existing package authorizations.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'2131'	
value	INSTANCE_OF	CHRSTRDR - Character String Data Representation
	MINLEN	1
	MAXLEN	255
	DFTVAL	''
	NOTE	Means that the default is the end-user name (identifier) of the requester that initiated the bind process.
	MINLVL	7
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

- |                 |   |
|-----------------|---|
| <b>insvar</b>   | <i>BGNBND</i> (on page 110)<br><i>CHRSTRDR</i> (on page 156)<br><i>REBIND</i> (on page 753) |
| <b>Semantic</b> | <i>PKGATHKP</i> (on page 579)<br><i>PKGATHRVK</i> (on page 584)<br><i>SQL</i> (on page 835) |

**NAME**

PKGRPLALW — Package Replacement Allowed

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'241F'**Length** \***Class** codpnt

Package Replacement Allowed (PKGRPLALW) specifies that if a relational database package currently exists having the same specified package name (in PKGNAMCT) and version name (PKGRPLVRS) as specified on the BGNBND command, then the existing package will be replaced if the current bind process results in a package being created. The VRSNAM parameter is specified on the BGNBND command and becomes the new package version name. The Bind Package Creation Control (BNDCRTCTL) parameter determines whether a package is created by the current bind process.

If a package is being replaced, see *PKGATHKP* (on page 579) and *PKGATHRVK* (on page 584) (values of the PKGATHOPT parameter) for explanations on how that affects existing package authorizations.

**SEE ALSO****insvar** *PKGRPLOPT* (on page 604)**Semantic** *BNDCHKONL* (on page 126)  
*PKGATHOPT* (on page 580)  
*PKGOWNID* (on page 600)  
*REBIND* (on page 753)



**NAME**

PKGRPLNA — Package Replacement Not Allowed

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2420'**Length** \***Class** codpnt

Package Replacement Not Allowed (PKGRPLNA) specifies that if a package currently exists that has the same package name and version name (VRSNAM) as specified on the BGNBND command, the bind process must not be initiated. A BGNBNDRM is returned preceding an SQLCARD reply data object that indicates the cause of the error.

**SEE ALSO**

**insvar**            *BGNBND* (on page 110)  
                    *PKGRPLOPT* (on page 604)  
                    *PKGRPLVRS* (on page 605)

**NAME**

PKGRPLOPT — Package Replacement Option

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'211C'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Package Replacement Option (PKGRPLOPT) String specifies whether the current bind process should replace an existing package that has the same package and version name as either of the following specify:

- The PKGNAMCT and VRSNAM parameters
- The PKGNAMCT and PKGRPLVRS parameter with the package and version name that the PKGNAMCT and VRSNAM parameters specify

If no package currently exists that has the same name, this parameter meaning and is ignored.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	6	
<b>class</b>	X'211C'	
<b>value</b>	INSTANCE_OF	CODPNTDR - Codepoint Data Representation
	ENUVAL	X'241F' - PKGRPLALW - Package Replacement Allowed
	ENUVAL	X'2420' - PKGRPLNA - Package Replacement Not Allowed
	DFTVAL	X'241F' - PKGRPLALW - Package Replacement Allowed
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

- insvar** *BGNBND* (on page 110)  
*PKGRPLVRS* (on page 605)
- Semantic** *BNDCHKONL* (on page 126)  
*PKGATHOPT* (on page 580)  
*PKGOWNID* (on page 600)  
*REBIND* (on page 753)

**NAME**

PKGRPLVRS — Replaced Package Version Name

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'212D'

**Length** \*

**Class** CLASS

**Sprcls** NAME - Name

Replaced Package Version Name (PKGRPLVRS) specifies the version name of the package being replaced by the package that the BGNBND command is binding.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'212D'	
value	INSTANCE_OF	NAMDR - Name Data
	MAXLEN	254
	DFTVAL	''
	NOTE	If a value is not specified for this parameter, the VRSNAM parameter on this command assumes its default value.
	NOTE	This parameter is ignored when PKGRPLOPT(PKGRPLNA) is specified.
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *BGNBND* (on page 110)

**Semantic** *PKGRPLALW* (on page 602)  
*PKGRPLOPT* (on page 604)

**NAME**

PKGSN — RDB Package Section Number

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'210C'

**Length** \*

**Class** CLASS

**Sprcls** FIELD - A Discrete Unit of Data

RDB Package Section Number (PKGSN) specifies a relational database package section number.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
sctnbr	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	16
	MINVAL	1
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>vldattls</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *CNTQRY* (on page 222)  
*EXCSQLSET* (on page 377)  
*OPNQRY* (on page 555)  
*PKGNAMECSN* (on page 596)

**Semantic** *CNTQRY* (on page 222)  
*OPNQRY* (on page 555)

**NAME**

PKGSNLST — RDB Package Name, Consistency Token, and Section Number List

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2139'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

RDB Package Name, Consistency Token, and Section Number List (PKGSNLST) specifies a list of fully qualified names of specific sections within one or more packages.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'2139'	
pkgnamcsn	INSTANCE_OF REPEATABLE	PKGNAMECSN - RDB Package Name, Consistency Token, and Section Number
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *RSLSETRM* (on page 785)

**Semantic** *LMTBLKPRC* (on page 475)

NAME

PKTOBJ — Packet Object

DESCRIPTION (Semantic)

**Dictionary** QDDBASD

**Codepoint** X'1C04'

**Length** \*

**Class** CODPNT

**Sprcls** SCALAR - Scalar Object

Packet Object (PKTOBJ) is used to send a packet from the source server to the target server and vice versa.

The content of a Packet Object is ignored.

<b>clsvar</b>	NIL	
<b>insvar</b>	INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'1C04'	
<b>value</b>	INSTANCE_OF	BYTSTRDR - Byte String
	MINLVL	7
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

SEE ALSO

**cmdtda** *SNDPKT* (on page 829)

**rpydta** *SNDPKT* (on page 829)

**Semantic** *RESPKTSZ* (on page 763)  
*SNDPKT* (on page 829)

**NAME**

PLGIN — Plug-in Security

**DESCRIPTION (Semantic)****Dictionary** QDDPRMD**Length** \***Class** CONSTANT

The Plug-in Security Mechanism specifies that an external authentication mechanism is to be used to carry out the authentication. The name of the plug-in to be used is negotiated during ACCSEC/ACCSECRD time and authentication tokens are exchanged at SECCHK/SECCHKRM. This security mechanism can be implemented using the Generic Security Service API (GSS-API) or an alternative interface that generates a similar sequence of security context information.

See the following terms for the specific meaning and function of each security mechanism component:

PLGINCNT	Security Plug-in List Count (see <i>PLGINCNT</i> (on page 610))
PLGINID	Security Plug-in-specific Identifier (see <i>PLGINID</i> (on page 611))
PLGINLSE	Security Plug-in List Entry (see <i>PLGINLSE</i> (on page 612))
PLGINLST	Security Plug-in List (see <i>PLGINLST</i> (on page 613))
PLGINNM	Security Plug-in Name (see <i>PLGINNM</i> (on page 614))
PLGINOVR	Plug-in Security Overview (see <i>PLGINOVR</i> (on page 615))
PLGINPPL	Security Plug-in Principal Name (see <i>PLGINPPL</i> (on page 619))
PLGINSECTKN	Plug-in Security Token (see <i>PLGINSECTKN</i> (on page 620))

**SEE ALSO**

<b>insvar</b>	<i>ACCSEC</i> (on page 52) <i>ACCSECRD</i> (on page 58) <i>PRCCNVCD</i> (on page 621) <i>SECCHK</i> (on page 800) <i>SECMEC</i> (on page 811)
<b>Semantic</b>	<i>ACCSEC</i> (on page 52) <i>PLGINOVR</i> (on page 615) <i>SECCHK</i> (on page 800) <i>SECMEC</i> (on page 811) <i>SECMGR</i> (on page 814)

**NAME**

PLGINCNT — Security Plug-in List Count

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'190F'

**Length** \*

**Class** CLASS

**Sprcls** SCALAR - Scalar Object

Security Plug-in List Count (PLGINCNT) is a count of the number of entries in the security plug-in list (PLGINLST).

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'190F'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	16
	MINVAL	1
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *PLGINLST* (on page 613)

**Semantic** *PLGIN* (on page 609)



**NAME**

PLGINID — Security Plug-in-specific Identifier

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'190D'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Security Plug-in-specific Identifier (PLGINID) specifies the release level of a security plug-in module.

This information is optional and may be reported by the plug-in module to help further identify the plug-in specified in PLGINNM.

The length may not be greater than 8 bytes with the format of the PLGINID being PPPVRRM, where:

*ppp* = A three-character product identifier

*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 plug-in)

*M* = *d*, where *d* is an integer and  $0 \leq d \leq 9$   
(0 means no modification level associated with the level of the plug-in)

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'190D'	
value	INSTANCE_OF	BYTSTRDR - Byte String
	MINLVL	7
	MAXLEN	8
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** ACCSEC (on page 52)  
 PLGINLSE (on page 612)  
 PRCCNVCD (on page 621)  
 SECCHK (on page 800)

**Semantic** PLGIN (on page 609)

**NAME**

PLGINLSE — Security Plug-in List Entry

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1910'

**Length** \*

**Class** CLASS

**Sprcls** ORDCOL - Ordered Collection

Security Plug-in List Entry (PLGINLSE) contains information for one security plug-in in the security plug-in list PLGINLST. This information consists of the name of the plug-in and, optionally, the service principal name that may be needed by the source server to initialize the security context and/or the plug-in version information.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'1910'	
plginid	INSTANCE_OF OPTIONAL	PLGINID - Security Plug-in-specific Identifier
plginnm	INSTANCE_OF REQUIRED	PLGINNM - Security Plug-in Name
plginppl	INSTANCE_OF OPTIONAL	PLGINPPL - Security Plug-in Principal Name
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *PLGINLST* (on page 613)

**Semantic** *PLGIN* (on page 609)

**NAME**

PLGINLST — Security Plug-in List

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'191E'

**Length** \*

**Class** CLASS

**Sprcls** ORDCOL - Ordered Collection

Security Plug-in List (PLGINLST) identifies the current set of authentication plug-in modules supported by the target server. The list must contain at least one entry. The ordering of the entries is significant in that the security service on the source server will use the first plug-in name in the list that it can support.

See *PLGINOVR* (on page 615) for information on the DDM flows that carry the PLGINLST.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'191E'	
plgincnt	INSTANCE_OF REQUIRED	PLGINCNT - Security Plug-in List Count
plginlse	INSTANCE_OF REPEATABLE REQUIRED	PLGINLSE - Security Plug-in List Entry
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

- insvar** ACCSECRD (on page 58)  
PRCCNVCD (on page 621)
- Semantic** ACCSEC (on page 52)  
PLGIN (on page 609)  
PLGINCNT (on page 610)  
PLGINLSE (on page 612)  
PLGINOVR (on page 615)

**NAME**

PLGINNM — Security Plug-in Name

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'191C'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Security Plug-in Name (PLGINNM) is used for carrying the name of the security plug-in to be used for authentication. The value is a character string in the CCSID of the CCSIDMGR negotiated at EXCSAT (or CCSID 500 if CCSIDMGR is not supported).

It is recommended that the name of the security plug-in is prefixed with the name of the vendor supplying it.

See *PLGINOVR* (on page 615) for information on the DDM flows that carry the PLGINNM.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'191C'	
value	INSTANCE_OF	CHRSTRDR - Character String
	MINLVL	7
	MINLEN	1
	MAXLEN	255
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *ACCSEC* (on page 52)  
*ACCSECRD* (on page 58)  
*PLGINLSE* (on page 612)  
*PRCCNVCD* (on page 621)  
*SECCHK* (on page 800)

**Semantic** *PLGIN* (on page 609)  
*PLGINID* (on page 611)  
*PLGINOVR* (on page 615)

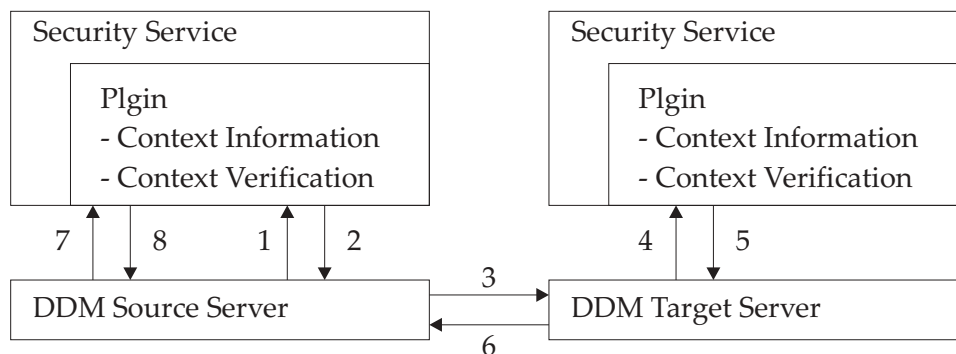
**NAME**

PLGINOVR — Plug-in Security Overview

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

The source and target servers may employ security services whose security architectures allow for pluggable modules to handle authentication processing. This provides the flexibility to use authentication methods beyond the current list of recognized DRDA security mechanisms. The Plug-in Security Overview (PLGINOVR) provides an overview of the DDM flows while using the Plug-in security mechanism.

Figure 3-65 (on page 616) indicates the DDM commands and replies that flow in the normal process of establishing a connection while using the Plug-in security mechanism for identification and authentication. Figure 3-64 shows an example of the exchange of messages to obtain and use the security context information within the security token. The DDM servers will interact with a Security Service which loads a Security Plug-in Module. Note that the Security Service at the target server may have multiple plug-in modules available to it but it will choose, where available, a module that is compatible with the module on the source server side. Compatibility requires that the name of the plug-in modules on both sides be identical and that they are able to interpret the security tokens received from one another.

**Figure 3-64** Example of Plug-In-Based Flows

The following is a brief description of the example plug-in flows shown in Figure 3-64:

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 service 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.

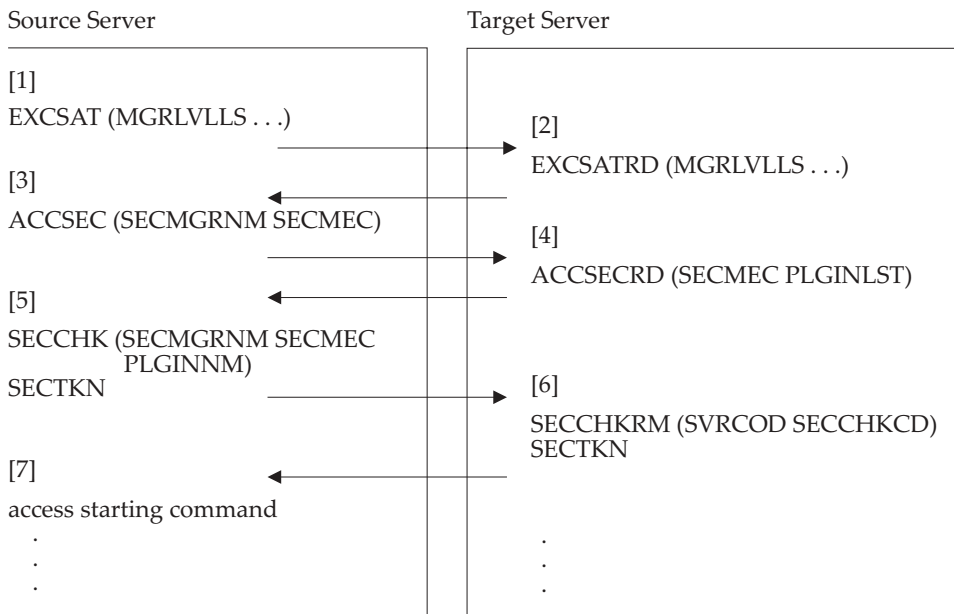


Figure 3-65 DDM Plug-In Security Flows

Figure 3-65 describes the synchronization of security information for a source server and target server pair. If a source server is connecting to multiple target servers, the security information shared between each source server and target server pair is independent from each of the other pairs. The following is a brief description of Figure 3-65 which shows some of the parameters for the DDM commands. See the appropriate terms for a detailed description of the parameters.

1. The source server identifies the level of security it would like to operate at by identifying the SECMGR at manager level 7 on the EXCSAT command.

```

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

2. The target 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 target server sends the EXCSATRD reply data to the source server.

```

EXCSATRD (
    MGRLVLLS (
    
```

```

MGRLVL (SECMGR, 7)
.
.
.))

```

3. The source server receives the EXCSATRD reply data. The type of identification and authentication mechanism is negotiated through the ACCSEC command. The SECMEC parameter indicates the security mechanism to use. This example assumes that the PLGIN security mechanism is specified but no PLGINNM has been provided in the ACCSEC.
4. The target server receives the ACCSEC command. It supports the security mechanism identified in the SECMEC parameter and returns the value in the ACCSECRD. Also in the ACCSECRD, the list of supported plug-in modules supported by the target server will be provided in the PLGINLST. Furthermore, these entries in the list will be ordered from the most preferred to the least preferred and there must be at least one entry.

If the source server had specified a PLGINNM in the ACCSEC, then if the target supported the specified plug-in module, then it would return this value as the single entry in the PLGINLST in addition to a SECMEC of PLGIN. If the target server did not support the specified plug-in module or if no plug-in module was specified, then it would return an ordered list of supported plug-in modules in the PLGINLST and there must be at least one entry.

If the target 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 target server but the PLGINNM is unspecified or unrecognized, the target server returns the security mechanism combination values that it does support in the SECMEC parameter in the ACCSECRD object. If the SECMEC includes PLGIN, then the ACCSECRD will 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 source server receives the ACCSECRD object and generates the security token containing the security context information required for security processing.

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

The source server passes the security context information in a SECCHK command with a SECTKN object. For information about the Plug-in security context information, see *PLGINSECTKN* (on page 620).

6. The target 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 target server may either process the security context information itself or it may call a security resource manager to process the security context information.

The target server generates a SECCHKRM to return to the source server. 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 target server or the security resource manager in conjunction with the failure, then it will be passed back to the source server in a SECTKN object.

7. The source server receives the SECCHKRM.

Assuming security processing is successful, the source server sends a data access starting command to the target 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 for processing, regardless of the SECCHKCD value.

If security processing fails, the source server might attempt recovery before returning to the application. For example, if the context information has expired, the source server could request new security context information to send to the target server. If the error condition is not recoverable, the source server returns to the application with an error indicating a security verification failure.

**SEE ALSO**

<b>insvar</b>	<i>SECCHK</i> (on page 800)
<b>Semantic</b>	<i>PLGIN</i> (on page 609)
	<i>PLGINLST</i> (on page 613)
	<i>PLGINNM</i> (on page 614)
	<i>PLGINPPL</i> (on page 619)
	<i>PLGINSECTKN</i> (on page 620)
	<i>SECCHK</i> (on page 800)
	<i>SECMEC</i> (on page 811)
	<i>SECMGR</i> (on page 814)
	<i>SECOVR</i> (on page 819)



**NAME**

PLGINPPL — Security Plug-in Principal Name

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1911'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Security Plug-in Principal Name (PLGINPPL) is used for carrying the service principal name of a plug-in module. The value is a character string in the CCSID of the CCSIDMGR negotiated at EXCSAT (or CCSID 500 if CCSIDMGR is not supported).

See *PLGINOVR* (on page 615) for information on the DDM flows that carry the PLGINPPL.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'1911'	
value	INSTANCE_OF	CHRSTRDR - Character String
	MINLVL	7
	MINLEN	1
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *PLGINLSE* (on page 612)

**Semantic** *PLGIN* (on page 609)

**NAME**

PLGINSECTKN — Plug-in Security Token

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Length** \*

**Class** HELP

The Plug-in Security Token (PLGINSECTKN) is information provided and used by the plug-in security mechanism. See *PLGINOVR* (on page 615) for information on the DDM flows that carry the PLGINSECTKN. The PLGINSECTKN is carried in a SECTKN object as command and reply data to the SECCHK command.

The security token contains plug-in security context information to identify and authenticate a user to the target server or to authenticate the target server to the source server.

The DDM architecture treats the PLGINSECTKN as a byte string. The SECMGR is aware of and has knowledge of the contents of the PLGINSECTKN. DDM does not know about the specific contents of this field. If the reader needs to know the actual contents, refer to *PLGINOVR*.

**SEE ALSO**

**Semantic** *PLGIN* (on page 609)  
*PLGINOVR* (on page 615)

**NAME**

PRCCNVCD — Conversation Protocol Error Code

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'113F'  
**Length** \*  
**Class** CLASS  
**Sprcls** STRING - String

Conversation Protocol Error Code (PRCCNVCD) String specifies the condition for which the PRCCNVRM was returned.

clsvar	NIL	
insvar	CLASS INSTANCE VARIABLES	
length	5	
class	X'113F'	
value	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	LENGTH	2
	ENUVAL	X'01' - RPYDSS received by target communications manager.
	ENUVAL	X'02' - Multiple DSSs sent without chaining or multiple DSS chains sent.
	ENUVAL	X'03' - ODJDSS sent when not allowed.
	ENUVAL	X'04' - Request correlation identifier of an RQSDSS is less than or equal to the previous RQSDSS request correlation identifier in the chain. If two RQSDSSs have the same request correlation identifier, the PRCCNVRM must be sent in a RPYDSS with a request correlation identifier of -1.
	ENUVAL	X'05' - Request correlation identifier of an OBJDSS does not equal the request correlation identifier of the preceding RQSDSS.
	ENUVAL	X'06' - EXCSAT was not the first command after the connection was established.
	ENUVAL	X'10' - ACCSEC or SECCHK command sent in wrong state.
	NOTE	See SECMGR for rules.
	MINLVL	5
ENUVAL	X'11' - SYNCCTL or SYNCRSY command is used incorrectly.	
NOTE	See SYNCPTOV for rules.	
MINLVL	5	

ENUVAL	X'12' - RDBNAM mismatch between ACCSEC, SECCHK, and ACCRDB.
MINLVL	5
ENUVAL	X'14'
MINLVL	7
NOTE	This value means that an incomplete scroll set is pending for a scrollable cursor, but the QRYROWSET value on the next CNTQRY did not match the number of rows pending in the rowset or QRYROWSET was not specified.
ENUVAL	X'15'
MINLVL	7
NOTE	This value means that an incomplete scroll set is pending for a scrollable cursor, but the QRYSCRORN/QRYROWNBR on the next CNTQRY did not default or resolve to a FETCH NEXT request.
ENUVAL	X'16'
MINLVL	7
NOTE	This value means that a QRYROWSET value was specified for a cursor with a QRYPRCTYP of LMTBLKPRC, even though the OPNQRY or the first CNTQRY for the cursor did not specify a QRYROWSET parameter.
ENUVAL	X'17'
MINLVL	7
NOTE	This value means that a QRYROWSET value was not specified for a cursor with a QRYPRCTYP of LMTBLKPRC, even though the OPNQRY or the first CNTQRY for the cursor did specify a QRYROWSET parameter.
ENUVAL	X'18'
MINLVL	7
NOTE	This value means that one or more scrolling parameters were received on a CNTQRY for a non-scrollable cursor.
ENUVAL	X'19'
MINLVL	7
NOTE	This value means that an RTNEXTDTA value of RTNEXTALL was not received on a CNTQRY command even though a QRYROWSET value was specified.
ENUVAL	X'1A'
MINLVL	7
NOTE	This value means that a QRYROWSET value was received on a CNTQRY command even though the QRYATTSNS value returned on the OPNQRYRM for the cursor was QRYSNSDYN.

ENUVAL	X'1C'
MINLVL	7
NOTE	The atomic EXCSQLSTT chain as initiated by a BGNATMCHN command contains an invalid command.
ENUVAL	X'1D'
MINLVL	7
NOTE	The ENDATMCHN command is sent without a prior matching BGNATMCHN command.
ENUVAL	X'1E'
MINLVL	7
NOTE	The value of the <i>nbrrow</i> instance variable on an EXCSQLSTT command for a multi-row input operation does not match the number of input data rows.
ENUVAL	X'20'
MINLVL	7
NOTE	Package name not provided. Default package name not established.
ENUVAL	X'21'
MINLVL	7
NOTE	This QRYROWSET value was not received on a CNTQRY for a rowset-enabled cursor or result set cursor.
ENUVAL	X'22'
MINLVL	7
NOTE	An override manager level in a MGRVLVLOVR is greater than the corresponding manager level negotiated between the requester and server by EXCSAT/EXCSATRD.
ENUVAL	X'23'
MINLVL	7
NOTE	One of the following security plug-in errors: <ol style="list-style-type: none"> <li>1. A PLGINNM was specified on an ACCSEC or SECCHK but the SECMEC did not include PLGIN.</li> <li>2. A PLGINLST was specified on an ACCSECRD, but the SECMEC did not include PLGIN.</li> <li>3. A PLGINID was specified on an ACCSEC or SECCHK but no PLGINNM was specified.</li> </ol>
ENUVAL	X'24'
MINLVL	7
NOTE	The DSS type is not Encrypted OBJDSS for security-sensitive data, even though the <i>secmec</i> negotiated between the requester and the server by ACCSEC/ACCSECRD requires encryption of

---

security-sensitive data.

---

<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

---

**SEE ALSO**

<b>insvar</b>	<i>BGNATMCHN</i> (on page 107) <i>ENDATMCHN</i> (on page 333) <i>EXCSQLSTT</i> (on page 381) <i>HEXSTRDR</i> (on page 429) <i>PRCCNVCD</i> (on page 621) <i>PRCCNVRM</i> (on page 625) <i>SYNERRC</i> (on page 985)
<b>Semantic</b>	<i>CLASS</i> (on page 158) <i>QDDRDBD</i> (on page 657) <i>STRING</i> (on page 888)

**NAME**

PRCCNVRM — Conversational Protocol Error

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1245'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

Conversational Protocol Error (PRCCNVRM) Reply Message indicates that a conversational protocol error occurred.

**DSS Carrier:** RPYDSS

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'1245'	
<b>prccnvcd</b>	INSTANCE_OF REQUIRED	PRCCNVCD - Conversational Protocol Error Code
<b>rdbnam</b>	INSTANCE_OF MINLVL OPTIONAL	RDBNAM - Relational Database Name 3
<b>recnt</b>	INSTANCE_OF MINVAL OPTIONAL NOTE	RECCNT - Record Count 0 Required for requests to insert multiple records in a file. Commands that operate on relational databases (RDBs) do not return this parameter.
	MINLVL	3
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL ENUVAL ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code 16 - SEVERE - Severe Error Severity Code 128 - SESDMG - Session Damage Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

## SEE ALSO

<b>cmdrpy</b>	<p> <i>ACCRDB</i> (on page 42)  <i>ACCSEC</i> (on page 52)  <i>BGNATMCHN</i> (on page 107)  <i>BGNBND</i> (on page 110)  <i>BNDSQLSTT</i> (on page 136)  <i>CLSQRY</i> (on page 165)  <i>CNTQRY</i> (on page 222)  <i>DRPPKG</i> (on page 293)  <i>DSCRDBTBL</i> (on page 300)  <i>DSCSQLSTT</i> (on page 304)  <i>ENDATMCHN</i> (on page 333)  <i>ENDBND</i> (on page 336)  <i>EXCSAT</i> (on page 363)  <i>EXCSQLIMM</i> (on page 371)  <i>EXCSQLSET</i> (on page 377)  <i>EXCSQLSTT</i> (on page 381)  <i>INTRDBRQS</i> (on page 445)  <i>OPNQRY</i> (on page 555)  <i>PRPSQLSTT</i> (on page 636)  <i>RDBCMM</i> (on page 728)  <i>RDBRLLBCK</i> (on page 745)  <i>REBIND</i> (on page 753)  <i>SECCHK</i> (on page 800)  <i>SYNCCTL</i> (on page 915)  <i>SYNCRSY</i> (on page 982) </p>
<b>insvar</b>	<p> <i>PRCCNVCD</i> (on page 621)  <i>SYNERRCD</i> (on page 985) </p>
<b>rpydta</b>	<p> <i>COMMAND</i> (on page 240) </p>
<b>Semantic</b>	<p> <i>CNTQRY</i> (on page 222)  <i>DSS</i> (on page 308)  <i>EXCSAT</i> (on page 363)  <i>EXCSQLSTT</i> (on page 381)  <i>OPNQRY</i> (on page 555)  <i>PRCCNVCD</i> (on page 621)  <i>QRYROWSET</i> (on page 697)  <i>SECMGR</i> (on page 814)  <i>SYNCPTOV</i> (on page 944) </p>



**NAME**

PRCNAM — Procedure Name

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'2138'

**Length** \*

**Class** CLASS

**Sprcls** MGRNAM - Manager Name

The Procedure Name (PRCNAM) Manager Name is an unarchitected string. DDM assumes that the user provides a name to the DDM source server that is formatted as the target server requires for locating the procedure name. The named string contains qualifiers for directories, libraries, catalogs, instances, or other levels of identification of the procedure name.

The target agent validates the procedure name according to its own rules for naming. This is done before an attempt to use the specified procedure name. This name must follow the target system's semantic and syntactic rules for procedure names.

The default value of PRCNAM is the procedure name value contained within the section identified by the *pkgnamcsn* parameter. If that value is null, then the *prcnam* parameter must be specified.

No semantic meaning is assigned to procedure names in DDM.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'2138'	
value	INSTANCE_OF OPTIONAL NOTE	NAMDR - Name Data  The prcnam is REQUIRED if the procedure name is specified by a host variable.
	DFTVAL NOTE	" Use the procedure name contained in the section specified by the <i>pkgnamcsn</i> parameter on the EXCSQLSTT command.
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *EXCSQLSTT* (on page 381)

**Semantic** *EXCSQLSTT* (on page 381)

NAME

PRCOVR — DDM Processing Overview

DESCRIPTION (Semantic)

Dictionary QDDTTRD

Length \*

Class HELP

DDM Processing Overview (PRCOVR) discusses how the DDM architecture provides local/remote transparency to local application programs using remote services. Figure 3-66 illustrates how an application program accesses a file as if all access method services and data management services are provided locally.

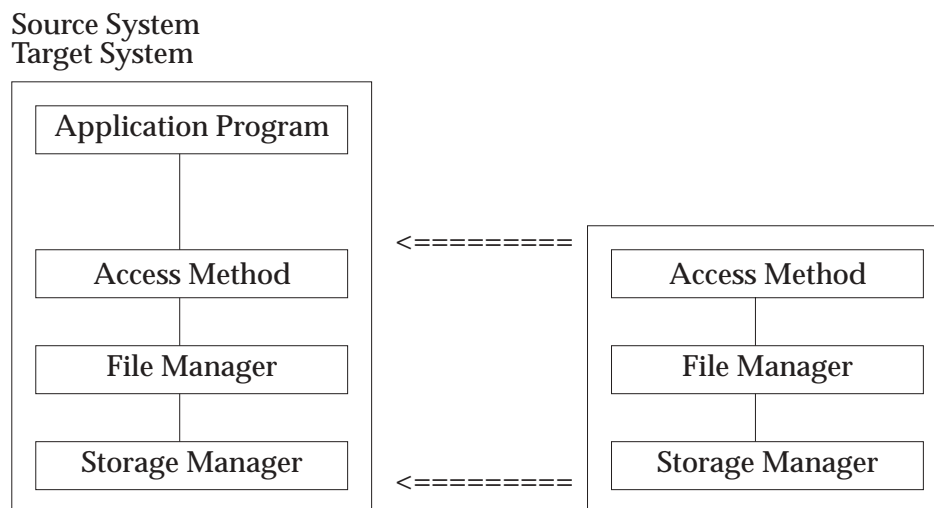
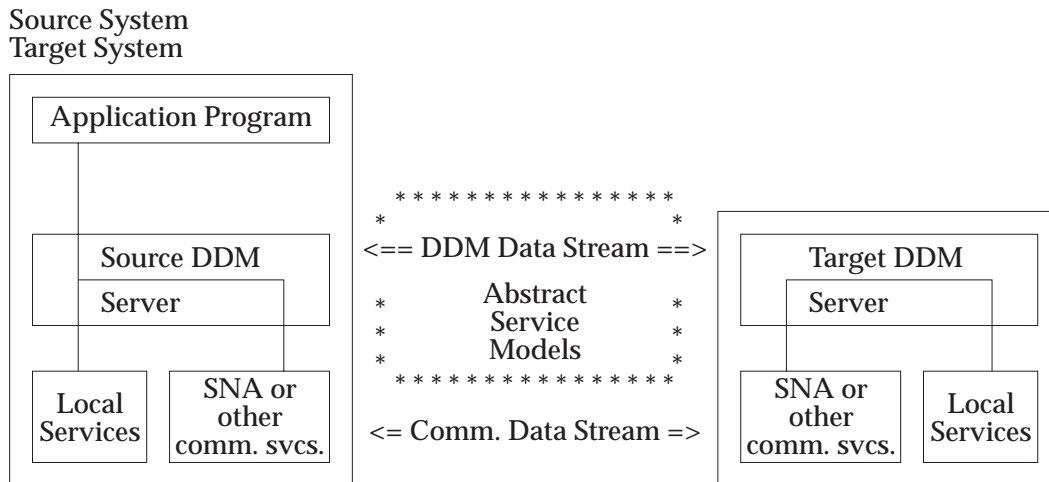


Figure 3-66 Transparent Data Management Services

To accomplish this, the DDM architecture is designed in terms of abstract services that the local system appears to perform. Figure 3-67 (on page 629) illustrates the actual processing DDM servers perform when remote services are required. An application program's request for a service is examined to determine if it can be satisfied locally or if it must be redirected to a remote server. If a remote service is required, the SOURCE DDM server creates a DDM message and transmits it to a remote TARGET DDM server. The target server responds by requesting the service from its local system. From the results of the service, a DDM message is created and returned to the source DDM server, where it is examined to provide a response to the application program.

All DDM messages are defined in terms of abstract service models. The source server does not attempt to use the services of the remote system directly since the interfaces and protocols for services differ from system to system. Instead, it requests services by using the commands that can be sent to each of the abstract service models of the DDM architecture. The target server is designed to respond to those commands the DDM architecture defines using the local services of the target system.



**Figure 3-67** DDM Processing Overview

Figure 3-67 illustrates the processing that occurs when executing a data management request to a remote file, but it does not adequately model that processing or the entities involved in it. Some of the subtasks that must be performed are:

1. Initiating communications
2. Negotiating connectivity
3. Locating resources
4. Checking authorization
5. Locking resources
6. Translating local interfaces to DDM commands
7. Translating DDM command to local interfaces
8. Controlling the flow of requests and replies
9. Converting data
10. Error handling and recovery
11. Terminating communications

The processing of these subtasks, but not their implementation, must also be standardized. Implementations of DDM are not required to support all commands defined for all objects by DDM architecture. See *SUBSETS* (on page 902) for a discussion of the subsetting rules of DDM architecture.

**SEE ALSO**

**Semantic**      *CONCEPTS* (on page 243)

**NAME**

PRDDTA — Product-specific Data

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2104'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Product-specific Data (PRDDTA) String specifies product-specific information that is conveyed to the target if the target's SRVCLSNM is not known when the ACCRDB command is issued. In addition, PRDDTA specifies that the data must be conveyed.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'2104'	
value	INSTANCE_OF MAXLEN REQUIRED	BYTSTRDR - Byte String 255
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** ACCRDB (on page 42)

**NAME**

PRDID — Product-specific Identifier

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'112E'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Product-specific Identifier (PRDID) specifies the product release level of a DDM server.

The contents of this parameter are unarchitected. No more than eight bytes can be sent.

PRDID should not be used in place of product-defined extensions to carry information not related to the product's release level.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
<b>length</b>	*	
<b>class</b>	X'112E'	
<b>value</b>	INSTANCE_OF MAXLEN REQUIRED	BYTSTRDR - Byte String 8
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar**            *ACCRDB* (on page 42)  
                       *ACCRDBRM* (on page 48)  
                       *RSCLMTRM* (on page 778)

**NAME**

PRMDMG — Permanent Damage Severity Code

**DESCRIPTION (Semantic)****Dictionary** QDDPRMD**Codepoint** X'002E'**Length** \***Class** CONSTANT

Permanent Damage Severity Code (PRMDMG) specifies that the state or value of the server's permanent objects has been damaged. Recovery from permanent damage may require special action that cannot be invoked through DDM commands, such as loading a backup file.

Further processing of the command depends on the architected specifications of the request, the permanent damage condition, and the environment of execution. For example, continued processing might be possible with other undamaged server resources.

---

value	64
-------	----

**SEE ALSO**

**insvar**            *AGNPRMRM* (on page 65)  
                       *CMDCHKRM* (on page 173)  
                       *RSCLMTRM* (on page 778)  
                       *SVRCOD* (on page 911)

**NAME**

PRMNSPRM — Parameter Not Supported

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1251'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

Parameter Not Supported (PRMNSPRM) Reply Message indicates that the specified parameter is not recognized or not supported for the specified command.

The PRMNSPRM can only be returned as the architected rules for DDM subsetting specify (see *SUBSETS* (on page 902)).

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'1251'	
<b>codpnt</b>	INSTANCE_OF REQUIRED NOTE	CODPNT - Codepoint  Specifies the codepoint of the parameter not supported.
<b>rdbnam</b>	INSTANCE_OF MINLVL OPTIONAL	RDBNAM - Relational Database Name 3
<b>svrdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code  8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

- cmdrpy**      *ACCRDB* (on page 42)  
                  *BGNBND* (on page 110)  
                  *BNDSQLSTT* (on page 136)  
                  *CNTQRY* (on page 222)  
                  *DRPPKG* (on page 293)  
                  *DSCRDBTBL* (on page 300)  
                  *DSCSQLSTT* (on page 304)  
                  *EXCSAT* (on page 363)  
                  *EXCSQLIMM* (on page 371)  
                  *EXCSQLSET* (on page 377)  
                  *EXCSQLSTT* (on page 381)

	<i>INTRDBRQS</i> (on page 445)
	<i>OPNQRY</i> (on page 555)
	<i>PRPSQLSTT</i> (on page 636)
	<i>RDBCMM</i> (on page 728)
	<i>RDBRLLBCK</i> (on page 745)
	<i>REBIND</i> (on page 753)
	<i>SECCHK</i> (on page 800)
	<i>SYNCCTL</i> (on page 915)
	<i>SYNCRSY</i> (on page 982)
<b>rpydta</b>	<i>COMMAND</i> (on page 240)
<b>Semantic</b>	<i>SUBSETS</i> (on page 902)



**NAME**

PRPHRCLST — List of Prepared and Heuristically Completed XIDs

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1905'

**Length** \*

**Class** CLASS

**Sprcls** COLLECTION

PRPHRCLST provides a list of XIDs which represents prepared or heuristically completed transactions at the target server.

An *xidcnt* value of zero indicates that no such transaction branches exist.

<b>clsvar</b>	NIL	
<b>insvar</b>	INSTANCE VARIABLES	
length	*	
class	X'1905'	
xid	INSTANCE_OF REQUIRED REPEATABLE NOTE	XID - Global Transaction Identifier  List of XIDs should not be specified if the <i>xidcnt</i> value is zero.
xidcnt	INSTANCE_OF REQUIRED NOTE	XIDCNT - XID List Count  A value of zero indicates the absence of an XID list.
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** SYNCCRD (on page 913)

**Semantic** XAMGROV (on page 1066)

## NAME

PRPSQLSTT — Prepare SQL Statement

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD

**Codepoint** X'200D'

**Length** \*

**Class** CLASS

**Sprcls** COMMAND - Command

The Prepare SQL Statement (PRPSQLSTT) command dynamically binds an SQL statement to a section in an existing relational database (RDB) package.

**Source System Processing**

The source system determines the location of the RDB:

Local Call the local RDB server.

Remote Send the PRPSQLSTT command to the remote RDB server.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**DSS Carrier: RQSDSS****Target System Processing**

The optional *bufinsind* parameter specifies whether and how the target server should optimize an SQL INSERT statement using the buffered insert technique when it gets executed as an atomic multi-row input operation against a partitioned database.

The optional *cmdsrcid* parameter uniquely identifies the source of the command.

The *monitor* parameter allows the requester to request a set of events or activities to be monitored on the target server. Refer to *MONITOR* (on page 521) for the list of supported events.

The *pkgnamcsn* parameter specifies the fully qualified package name, the package's consistency token, and a section number within the package in which the SQL statement is being prepared.

The *rdbnam* parameter specifies the name of the RDB which the ACCRDB command accessed. If the *rdbnam* parameter is specified, its value must be the same as the VALUE specified on the ACCRDB command for RDBNAM.

The TYPDEFNAM command data object specifies the name of the data type to data representation mapping definitions that the source SQLAM uses when sending command data objects for this command.

The TYPDEFOVR command data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type to data representation mapping definitions for the command data objects for this command.

The SQL statement being prepared is placed in the SQLSTT command data object and sent with the command. The SQLSTT FD:OCA descriptor describes the SQLSTT command data object.

Once the SQL statement has been prepared, it is executed by issuing the EXCSQLSTT command. This is possible until the unit of work, in which the PRPSQLSTT command was issued, ends.

The SECTKNOVR command data object is sent by the intermediate server, if the SQLSTT and optional SQLATTR command data objects are encrypted. The SECTKNOVR contains the encryption seed and the encryption token used to encrypt the SQLSTT and SQLATTR objects. The SECTKNOVR DSS is encrypted and must be sent prior to the encrypted SQLSTT command data object.

Normal completion of this command results in either an SQLCARD or an SQLDARD object being returned. The SQLCA FD:OCA descriptor describes the SQLCARD object. The SQLDARD FD:OCA descriptor describes the SQLDARD reply data object.

The SECTKNOVR reply data object is sent by the intermediate server, if the SQLDARD reply data object is encrypted. The SECTKNOVR contains the encryption seed and the encryption token used to encrypt the SQLDARD object. The SECTKNOVR DSS is encrypted and must be sent prior to the encrypted SQLDARD reply data object.

The TYPDEFNAM reply data object specifies the name of the data type to data representation mapping definitions that the target SQLAM uses when sending the reply data objects for this command.

The TYPDEFOVR reply data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type to data representation mapping definitions for the reply data objects for this command.

The RTNSQLDA instance variable controls whether to return an SQLDA descriptor area for the output variables for the prepared SQL statement.

The TYPSQLDA instance variable determines the type of descriptive information to be returned for the output variables. Three types of SQL descriptor area can be requested: a light, standard, or extended version. Each type provides a different set 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 API such as JDBC.

The SQLDA is defined by the SQLDARD FD:OCA descriptor. Refer to the DRDA Reference for a description of an SQLDARD.

### Exceptions

If the current unit of work ended abnormally, an ABNUOWRM reply message followed by an SQLCARD reply data object must be returned.

Otherwise, exception conditions the RDB detects are reported in the SQLDARD or SQLCARD reply data object as follows, based on the value of the *rtnsqlda* parameter as specified (implicitly or explicitly) on the PRPSQLSTT command:

- *rtnsqlda=true*

The target server may return an SQLDARD or SQLCARD reply data object. If an SQLDARD reply data object is to be returned, the number of data variable definition entries contained therein must be zero. If instead an SQLCARD reply data object is returned, it can optionally be preceded by an SQLERRRM reply message.

- *rtnsqlda=false*

The target server must return an SQLCARD reply data object which can optionally be preceded by an SQLERRRM reply message.

If the target SQLAM detects that the value of the SQLSTT command data object does not have the characteristics the FD:OCA descriptor claims, then the command is rejected with the DTAMCHRM.

If the binding process is active, then the command is rejected with the PKGBPARM.

If the specified RDB is not currently accessed, then the command is rejected with the RDBNACRM.

**Source System Reply Processing**

Return any reply messages and data objects to the requester.

**Package Name and Consistency Token**

With SQLAM level 7, the PKGNAMCSN is changed to an optional instance variable. If specified, the values are used as the default package name and consistency token for the connection. If not specified on the command, the PKGSN is required to identify the section number. In this case, the default package name, consistency token, and specified section number is used for the request. If PKGNAMCSN was never specified on the connection, a PRCCNVRM is returned.

clsvar	NIL	
insvar	CLASS INSTANCE VARIABLES	
length	*	
class	X'200D'	
bufinsind	INSTANCE_OF OPTIONAL IGNORABLE	BUFINSIND - Buffered Insert Indicator  If the SQL statement is not an INSERT, or if the statement is not executed as an atomic multi-row input operation, or if the target database is not partitioned.
	MINLVL	7
cmdsrcid	INSTANCE_OF OPTIONAL MINLVL	CMDSRCID - Command Source Identifier  7
monitor	INSTANCE_OF OPTIONAL MINLVL	MONITOR - Monitor Events  7
pkgnamcsn	INSTANCE_OF  MINLVL OPTIONAL MTLEXC	PKGNAMCSN - RDB Package Name, Consistency Token, and Section Number  7 Required if PKGSN not specified. PKGSN
pkgsn	INSTANCE_OF MINLVL OPTIONAL	PKGSN - RDB Package Section Number 7 Required if PKGNAMCSN not specified.

	MTLEXC	PKGNAMECSN
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
rtnsqlda	INSTANCE_OF OPTIONAL	RTNSQLDA - Return SQL Descriptor Area
typsqlda	INSTANCE_OF MINLVL OPTIONAL NOTE ENUVAL NOTE ENUVAL NOTE ENUVAL NOTE DFTVAL	TYPSQLDA - Type of Descriptor to Return 7 Supported values: 0 Request standard output SQLDA. 2 Request light output SQLDA. 4 Return extended output SQLDA. 0
clscmd	NIL	
inscmd	NIL	
cmddda		<b>COMMAND OBJECTS</b>
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL MINLVL ENUVAL MINLVL OPTIONAL DFTVAL NOTE	TYPDEFNAM - Data Type Definition Name 'QTDSQL370' 'QTDSQL400' 'QTDSQLX86' 'QTDSQLASC' 4 'QTDSQLVAX' 4 " The default means the value received on ACCRDB command is used.
X'0035'	INSTANCE_OF OPTIONAL DFTVAL NOTE	TYPDEFOVR - TYPDEF Overrides " The default means the value received on ACCRDB command is used.
X'190B'	INSTANCE_OF OPTIONAL NOTE	SECTKNOVR - SECTKN Override This is sent by the intermediate server, if the SQLSTT and optional SQLATTR command data objects are encrypted. This must precede the SQLSTT command data object.
X'2414'	INSTANCE_OF REQUIRED	SQLSTT - SQL Statement
X'2450'	INSTANCE_OF	SQLATTR - SQL Statement Attributes

	OPTIONAL MINLVL	7
<b>rpydta</b>		<b>REPLY OBJECTS</b>
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL MINLVL ENUVAL MINLVL OPTIONAL DFTVAL NOTE	TYPDEFNAM - Data Type Definition Name 'QTDSQL370' 'QTDSQL400' 'QTDSQLX86' 'QTDSQLASC' 4 'QTDSQLVAX' 4 " The default means the value received on ACCRDBRM is used.
X'0035'	INSTANCE_OF OPTIONAL DFTVAL NOTE	TYPDEFOVR - TYPDEF Overrides " The default means the value received on ACCRDBRM is used.
X'190B'	INSTANCE_OF OPTIONAL NOTE	SECTKNOVR - SECTKN Override This is sent by the intermediate server, if the SQLDARD reply data object is encrypted. This must precede the SQLDARD reply data object.
X'1C00'	INSTANCE_OF OPTIONAL MINLVL	MONITORRD - Monitor Reply Data 7
X'1C03'	INSTANCE_OF OPTIONAL MINLVL	MGRVLVLOVR - Manager Level Overrides 7
X'2411'	INSTANCE_OF REQUIRED MTLEXC NOTE	SQLDARD - SQLDA Reply Data X'2408' - SQLCARD - SQL Communications Area Reply Data Returned if RTNSQLDA (TRUE) is specified.
X'2408'	INSTANCE_OF MTLEXC	SQLCARD - SQL Communications Area Reply Data X'2411' - SQLDARD - SQLDA Reply Data
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'220D'	INSTANCE_OF	ABNUOWRM - Abnormal End Unit of Work Condition
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported

X'220E'	INSTANCE_OF	DTAMCHRM - Data Descriptor Mismatch
X'1253'	INSTANCE_OF	OBJNSPRM - Object Not Supported
X'2209'	INSTANCE_OF	PKGBPARAM - RDB Package Binding Process Active
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'2204'	INSTANCE_OF	RDBNACRM - RDB Not Accessed
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limits Reached
X'2213'	INSTANCE_OF	SQLERRRM - SQL Error Condition
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'125F'	INSTANCE_OF	TRGNSPRM - Target Not Supported
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

**SEE ALSO**

<b>inscmd</b>	<i>SQLAM</i> (on page 847)
<b>Semantic</b>	<i>BGNBND</i> (on page 110)
	<i>CMDSRCID</i> (on page 178)
	<i>PRPSQLSTT</i> (on page 636)
	<i>SECTKNOVR</i> (on page 822)
	<i>SQL</i> (on page 835)
	<i>SQLAM</i> (on page 847)

NAME

PRPSTTKP — Prepared Statement Keep

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
 X'2167'  
**Length** \*  
**Class** HEXSTRDR  
**Sprcls** STRING - String  
**Clsvar** NIL

Prepared Statement Keep (PRPSTTKP) 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. If this option is not specified, prepared statements are released when the work associated with it is committed or rolled back.

Exceptions

Exception conditions that the RDB detects are reported in a VALNSPRM (Parameter Value Not Supported) object.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	5	
class	X'2167'	
value	INSTANCE_OF	HEXSTRDR
	LENGTH	2
	ENUVAL	X'F0'
	NOTE	Dynamic statements are dropped during commit processing and during rollback processing.
	ENUVAL	X'F1'
	NOTE	Dynamic statements are kept during commit processing and may be used on subsequent requests, but are dropped during rollback processing.
	ENUVAL	X'F2'
	NOTE	Dynamic statements are dropped during commit processing, but are kept during rollback processing and may be used on subsequent requests.
	ENUVAL	X'F3'
	NOTE	Dynamic statements are kept during commit processing and during rollback processing, and may be used on subsequent requests.
	MINLVL	7
	DFTVAL	X'F0'



**SEE ALSO**

<b>cmdtta</b>	<i>BGNBND</i> (on page 110)
<b>rpydta</b>	<i>VALNSPRM</i> (on page 1057)

## NAME

PWDENC — Password Encryption Security Mechanism

## DESCRIPTION (Semantic)

**Dictionary** QDDTTRD

**Length** \*

**Class** HELP

The Password Encryption Security Mechanism (PWDENC) specifies the use of an encrypted password. See *USRSECOVR* (on page 1050) for more information about this mechanism.

The mechanism authenticates the user like the user ID and password mechanism, but the password is encrypted using 56-bit DES. The DES encryption seed is generated by exchanging connection keys in the SECTKN instance variable on the ACCSEC command and the ACCSEC reply data using the standard Diffie-Hellman key distribution algorithm to generate a shared private key. The application requester encrypts the password using the shared private key generated and the user ID. The user ID is passed in the USRID instance variable and the encrypted password is passed in the SECTKN instance variable on the SECCHK command. The application server decrypts the password using the shared private key and the user ID. The user ID and password are then passed to the local security user to be authenticated.

**Generating the Shared Private Key**

Diffie-Hellman algorithm is used to generate a shared private key between the application requester and application server. This secret key is used as the encryption seed to the DES encryption algorithm. Diffie-Hellman is the first standard public key algorithm ever invented. It gets its security from the difficulty of calculating discrete logarithms in a finite field. Diffie-Hellman requires three agreed upon values  $n$  and  $g$  such that  $g$  is a primitive of large prime  $n$  and the size of the exponent. These values are fixed. First, the application requester chooses a random large integer  $x$  and generates the value  $X$  where  $X=g^x \bmod n$ .  $X$  is sent in the SECTKN on the ACCSEC command. Second, the application server chooses another random large integer  $y$  and generates the value  $Y$  where  $Y=g^y \bmod n$ .  $Y$  is sent in the SECTKN on the reply to the ACCSEC command. The application requester computes the shared private key,  $k=Y^x \bmod n$ . The application server computes the shared private key,  $k1=X^y \bmod n$ . The middle 8-bytes of the 32 byte  $k$  is used as the DES encryption key.

Following are DRDA's Diffie-Hellman agreed public values for  $g$  (generator),  $n$  (prime), and the size of the exponent ( $x$  and  $y$ ). These values must be used as is to generate a shared private key:

```
static unsigned int prime_len = 32;
static unsigned char prime[32] = {
0xc6, 0x21, 0x12, 0xd7, 0x3e, 0xe6, 0x13, 0xf0,
0x94, 0x7a, 0xb3, 0x1f, 0x0f, 0x68, 0x46, 0xa1,
0xbf, 0xf5, 0xb3, 0xa4, 0xca, 0x0d, 0x60, 0xbc,
0x1e, 0x4c, 0x7a, 0x0d, 0x8c, 0x16, 0xb3, 0xe3,
};

static unsigned int generator_len = 32;
static unsigned char generator[32] = {
0x46, 0x90, 0xfa, 0x1f, 0x7b, 0x9e, 0x1d, 0x44,
0x42, 0xc8, 0x6c, 0x91, 0x14, 0x60, 0x3f, 0xde,
0xcf, 0x07, 0x1e, 0xdc, 0xec, 0x5f, 0x62, 0x6e,
0x21, 0xe2, 0x56, 0xae, 0xd9, 0xea, 0x34, 0xe4
};
```

```
static unsigned int exponent_bits = 255;
```

### **DES Password Encryption**

The Data Encryption Standard (DES), known as the Data Encryption Algorithm (DEA) by ANSI and the DEA-1 by ISO, is the worldwide standard for encrypting passwords. DES is a block cipher; it encrypts data in 64-bit blocks. DRDA password encryption uses DES CBC mode as defined by the FIPS standard (FIPS PUB 81). DES CBC requires an encryption key and an 8-byte user ID to encrypt the password. The key length is 56 bits. The Diffie-Hellman shared private key is 256 bits. To reduce the number of bits, 64 bits are selected from the connection key by selecting the middle 8 bytes and parity is added to the lower order bit of each byte producing a 56-bit key with 8 bits of parity. The user ID used is the value in the USRID instance variable passed in the SECCHK command. The USRID is zero-padded to 8 bytes if less than 8 bytes or truncated to 8 bytes if greater than 8 bytes.

To decrypt the encrypted password, DES CBC mode requires the zero-padded user ID, and the encryption key. Once the password is decrypted, the server uses the local security manager to validate the user ID/password combination.

### **SEE ALSO**

**Semantic**        *SECMEC* (on page 811)  
                  *USRENCPWD* (on page 1043)  
                  *USRSECOVR* (on page 1050)

**NAME**

PWDSBS — Password Substitution Security Mechanism

**DESCRIPTION (Semantic)**

**Dictionary** QDDTTRD

**Length** \*

**Class** HELP

The Password Substitution Security Mechanism (PWDSBS) specifies the use of a password substitute. See *USRSECOVR* (on page 1050) for more information about this mechanism.

The mechanism authenticates the user like the user ID and password mechanism, but the 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 Substitute Generation Summary**

1. Padded\_PW = Left justified PW padded to the right with X'40's to 8 bytes
2. XOR\_PW = Padded\_PW xor X'5555555555555555'
3. SHIFT\_RESULT = XOR\_PW shifted left 1 bit
4. PW\_TOKEN = ENCdes(SHIFT\_RESULT, ID)
5. RDrSEQ = RDr + 1
6. PW\_SUB = MACdes(PW\_TOKEN, (RDrSEQ,RDs,ID xor RDrSEQ,PWSEQs))

The terms used in the above summary are defined as follows:

xor	Exclusive or operation
PW	User password
ID	User identifier
ENCdes	Encipher using the Data Encryption Standard algorithm (the first parameter is the key)
MACdes	MAC (Message Authentication Code) using the Data Encryption Standard algorithm (the first parameter is the key)
Rds	Random data (application requester's seed) sent to the DRDA application server on ACCSEC
Rdr	Random data (application server's seed) received from the application server on ACCSECRD
PWSEQs	Sequence number for password substitution on the send side (always 1 for generation of a password substitute for DRDA and TCP/IP)
RDrSEQ	The arithmetic sum of RDr and the value of PWSEQs (1)

### Password Substitute Generation Details

The above summary illustrates the logic involved in generating a password substitute. The numbers in the following discussion correspond to the numbers in the summary.

To generate a password substitute several steps are required:

1. The user's password must be left-justified in an eight-byte variable and padded to the right with X'40' up to an 8-byte length. If the user's password is 8 bytes in length, no padding would occur. For computing password substitutes for passwords of length 9 and 10 see **Handling Passwords of Length 9 and 10** (on page 648). Passwords less than 1 byte or greater than 10 bytes in length are not valid.
2. The padded password is exclusive or'ed with 8 bytes of X'55'.
3. The entire byte result is shifted 1 bit to the left; the leftmost bit value is discarded and the rightmost bit value is cleared to 0.
4. This shifted result is used as key to the DES to encipher the user identifier. When the user identifier is less than 8 bytes, it is left justified in an 8 byte variable and padded to the right with X'40's. When the user identifier is 9 or 10 bytes, it is first padded to the right with X'40' to a length of 10 bytes. Then 3 bytes 9 and 10 are this algorithm in handling user IDs with lengths of 9 and 10 characters).

Byte 1, bits 0 and 1 are replaced with byte 1, bits 0 and 1 exclusive or'ed with byte 9, bits 0 and 1.

Byte 2, bits 0 and 1 are replaced with byte 2, bits 0 and 1 exclusive or'ed with byte 9, bits 2 and 3.

Byte 3, bits 0 and 1 are replaced with byte 3, bits 0 and 1 exclusive or'ed with byte 9, bits 4 and 5.

Byte 4, bits 0 and 1 are replaced with byte 4, bits 0 and 1 exclusive or'ed with byte 9, bits 6 and 7.

Byte 5, bits 0 and 1 are replaced with byte 5, bits 0 and 1 exclusive or'ed with byte 10, bits 0 and 1.

Byte 6, bits 0 and 1 are replaced with byte 6, bits 0 and 1 exclusive or'ed with byte 10, bits 2 and 3.

Byte 7, bits 0 and 1 are replaced with byte 7, bits 0 and 1 exclusive or'ed with byte 10, bits 4 and 5.

Byte 8, bits 0 and 1 are replaced with byte 8, bits 0 and 1 exclusive or'ed with byte 10, bits 6 and 7.

The result of this encryption is known as PW\_TOKEN in this discussion.

#### Notes:

1. DES refers to the Data Encryption Standard, Federal Information Processing Standards Publication 46, Data Encryption Standard, January 15, 1977.
2. Bit 0 is the high order bit (that is, has value of X'80').
3. User identifiers greater than 10 bytes or less than 1 byte in length are not valid.
5. A value of 1 is added to RDr, the random seed value received from the DRDA application server yielding RDrSEQ. (This operation is a vestige of the algorithm's original use in APPC communications.)

6. The PW\_TOKEN is used as a key to the DES MAC function to generate a MAC for the following string of values:
- RDrSEQ, the random data value (seed) received from the DRDA application server plus the sequence number of 1.
  - RDs, the random data value (seed) sent to the DRDA application server on the ACCSEC command.
  - A sixteen-byte value created by:
    - Padding the user identifier with x'40' to a length of 16 bytes.
    - Exclusive or'ing the two eight-byte halves of the padded user identifier with the RDrSEQ value (the first 8 bytes are exclusive or'ed, and then the second 8 bytes).
  - PWSEQs, the sequence number which in this step is always 1 for DRDA TCP/IP usage.

The resulting enciphered random data is the *password substitute*.

Because the user identifier-password combination is unique to the user and the random data changes randomly from connection to connection, the password substitute is valid only for a specific user's connection request (ACCSEC/SECCHK/ACCRDB).

#### Handling Passwords of Length 9 and 10

First generate PW\_TOKENa by using characters 1 to 8 of the password and steps 1-4 from **Password Substitute Generation Summary** (on page 646).

Second generate PW\_TOKENb by using characters 9 and 10 of the password and steps 1 to 4 from **Password Substitute Generation Summary** (on page 646). (In this case the Padded\_PW from step 1 will be characters 9 and 10 padded to the right with X'40' for a total length of 8.)

Set PW\_TOKEN to PW\_TOKENa exclusive or'ed with PW\_TOKENb.

Now compute the password substitute by performing steps 5 to 8 from **Password Substitute Generation Summary** (on page 646).

#### Verifying a Password Substitute

In order to verify a password substitute on the receive side, the DRDA application server must regenerate the password substitute and compare the results. There is no way to unwind the DES MAC function used to generate the password substitute.

Of course, opposite values must be used in generating the password substitute on the receive side, as follows:

- Where RDr is used on the send side, RDs must be used on the receive side.
- Where RDs is used on the send side, RDr must be used on the receive side.

A value of 1 is used for the sequence number as at the application requester.

#### SEE ALSO

<b>Semantic</b>	<i>SECMEC</i> (on page 811)
	<i>USRSBSPWD</i> (on page 1049)
	<i>USRSECOVR</i> (on page 1050)

**NAME**

PWDSEC — Password Security Mechanism

**DESCRIPTION (Semantic)**

**Dictionary** QDDTTRD

**Length** \*

**Class** HELP

The Password Security Mechanism (PWDSEC) specifies the use of the password security mechanism. See *USRSECOVR* (on page 1050) for more information about the password security mechanism.

**SEE ALSO**

**Semantic**        *SECMEC* (on page 811)  
                  *USRIDNWPWD* (on page 1045)  
                  *USRIDPWD* (on page 1047)  
                  *USRSBSPWD* (on page 1049)

**NAME**

PWDSSB — Strong Password Substitution Security Mechanism

**DESCRIPTION (Semantic)**

**Dictionary** QDDTTRD

**Length** \*

**Class** HELP

The Strong Password Substitution Security Mechanism (PWDSSB) specifies the use of a strong password substitute. See *USRSECOVR* (on page 1050) for more information about this mechanism.

The mechanism authenticates the user like the user ID and password mechanism, but the password does not flow. A password substitute is generated using the SHA-1 algorithm (see below), and is sent to the application server. The application server generates a password substitute using the same algorithm and compares it with the application requester's password substitute. If equal, the user is authenticated.

**Strong Password Substitute Generation Algorithm**

The client and server generate random seeds and exchange them. That is, the client generates a "random client seed" (RDs) and sends it to the server. Similarly, the server generates a "random server seed" (RDr) and sends it to the client. The random seeds are 8 bytes.

The client generates a 20-byte password token (PW\_TOKEN) as follows:

$$PW\_TOKEN = SHA-1(PW, ID)$$

The password (PW) and user name (ID) can be of any length greater than or equal to 1 byte.

The client generates a 20-byte password substitute (PW\_SUB) as follows:

$$PW\_SUB = SHA-1(PW\_TOKEN, RDr, RDs, ID, PWSEQs)$$

where PWSEQs is an 8-byte value which has a constant value of 0x0000000000000001.

The client sends the ID and PW\_SUB to the server.

**Note:** The PW\_SUB value sent in the SECTKN parameter is 20 bytes long here as opposed the 8-byte value sent with the original password substitute algorithm described in PWDSSB.

The server uses the same algorithm to create its own PW\_SUB which it compares to the one received from the client to verify the password.

**Note:** The SHA-1 algorithm is documented in the FIPS Secure Hash Standard (FIPS PUB 180-1).

**SEE ALSO**

None.



**NAME**

QDDBASD — DDM Base Classes Dictionary

**DESCRIPTION (Semantic)**

**Dictionary** QDDSRVDR

**Length** \*

**Class** DICTIONARY

This dictionary contains all DDM classes that describe the data objects required for DDM.

<b>mgrlvl</b>	1	
<b>mgrdepls</b>	INSTANCE_OF MGRVLVN NOTE	QDDPRMD - DDM Primitive Classes Dictionary 1 The primitives dictionary includes class descriptions for many classes on which QDDBASD classes depend.
<b>dctind</b>	*	
<b>objlst</b>	*	

**SEE ALSO**

- insvar** *INTTKNRM* (on page 448)
- mgrdepls** *QDDRDBD* (on page 657)
- Semantic**
  - ACCSEC* (on page 52)
  - ACCSECRD* (on page 58)
  - AGENT* (on page 61)
  - AGNPRMRM* (on page 65)
  - BGNATMCHN* (on page 107)
  - CCSIDDBC* (on page 148)
  - CCSIDMBC* (on page 149)
  - CCSIDMGR* (on page 150)
  - CCSIDSBC* (on page 154)
  - CMDATHRM* (on page 171)
  - CMDCHKRM* (on page 173)
  - CMDCMPRM* (on page 175)
  - CMDNSPRM* (on page 176)
  - CMNAPPC* (on page 184)
  - CMNMGR* (on page 196)
  - CMNSYNCPT* (on page 202)
  - CMNTCPIP* (on page 214)
  - CODPNTDR* (on page 235)
  - DCTIND* (on page 258)
  - DCTINDEN* (on page 259)
  - DEPERRC* (on page 270)
  - DICTIONARY* (on page 286)
  - DSSFMT* (on page 318)
  - ENCALG* (on page 331)
  - ENCKEYLEN* (on page 332)
  - ENDATMCHN* (on page 333)
  - ENDCHNTYP* (on page 341)
  - EXCSAT* (on page 363)

*EXCSATRD* (on page 369)  
*EXTNAM* (on page 402)  
*FDODTA* (on page 408)  
*FDOEXT* (on page 410)  
*FDOOBJ* (on page 411)  
*FDOOFF* (on page 412)  
*FORGET* (on page 423)  
*INTRDBRQS* (on page 445)  
*INTTKNRM* (on page 448)  
*IPADDR* (on page 450)  
*KERSECPPL* (on page 470)  
*LOGNAME* (on page 484)  
*LOGTSTMP* (on page 485)  
*MANAGER* (on page 491)  
*MGRDEPRM* (on page 505)  
*MGRLVL* (on page 506)  
*MGRLVLLS* (on page 508)  
*MGRLVLN* (on page 509)  
*MGRLVLOVR* (on page 511)  
*MGRLVLRM* (on page 512)  
*MGRNAM* (on page 513)  
*NEWPASSWORD* (on page 531)  
*OBJDSS* (on page 536)  
*OBJNSPRM* (on page 542)  
*PASSWORD* (on page 578)  
*PKGDFTC* (on page 589)  
*PKGRPLVRS* (on page 605)  
*PKTOBJ* (on page 608)  
*PLGINCNT* (on page 610)  
*PLGINID* (on page 611)  
*PLGINLSE* (on page 612)  
*PLGINLST* (on page 613)  
*PLGINNM* (on page 614)  
*PLGINPPL* (on page 619)  
*PLGINSECTKN* (on page 620)  
*PRCCNVRM* (on page 625)  
*PRCNAM* (on page 627)  
*PRDID* (on page 631)  
*PRMNSPRM* (on page 633)  
*PRPHRCLST* (on page 635)  
*RDBCMTOK* (on page 731)  
*RESPKTSZ* (on page 763)  
*RLSCONV* (on page 765)  
*RPYDSS* (on page 766)  
*RPYMSG* (on page 770)  
*RQSCRR* (on page 772)  
*RQSDSS* (on page 774)  
*RSCLMTRM* (on page 778)  
*RSCNAM* (on page 781)  
*RSCTYP* (on page 782)  
*RSNCOD* (on page 786)  
*RSYNCMGR* (on page 787)

*RSYNCTYP* (on page 790)  
*SECCHK* (on page 800)  
*SECCHKCD* (on page 806)  
*SECCHKRM* (on page 809)  
*SECMEC* (on page 811)  
*SECMGR* (on page 814)  
*SECMGRNM* (on page 818)  
*SECTKN* (on page 820)  
*SECTKNOVR* (on page 822)  
*SERVER* (on page 824)  
*SNAADDR* (on page 827)  
*SNDPKT* (on page 829)  
*SPVNAM* (on page 834)  
*SRVCLSNM* (on page 872)  
*SRVDGN* (on page 873)  
*SRVNAM* (on page 880)  
*SRVRLSLV* (on page 884)  
*STGLMT* (on page 885)  
*SUBSETS* (on page 902)  
*SUPERVISOR* (on page 908)  
*SVCERRNO* (on page 910)  
*SVRCOD* (on page 911)  
*SYNCCRD* (on page 913)  
*SYNCCTL* (on page 915)  
*SYNCLOG* (on page 922)  
*SYNCPTMGR* (on page 939)  
*SYNCRRD* (on page 981)  
*SYNCRSY* (on page 982)  
*SYNCTYPE* (on page 984)  
*SYNERRCD* (on page 985)  
*SYNTAXRM* (on page 989)  
*TCPHOST* (on page 999)  
*TCPPTHOST* (on page 1006)  
*TEXT* (on page 1018)  
*TIMEOUT* (on page 1019)  
*TRGNSPRM* (on page 1022)  
*UNORDERED* (on page 1036)  
*UOWID* (on page 1038)  
*UOWSTATE* (on page 1040)  
*USRID* (on page 1044)  
*VALNSPRM* (on page 1057)  
*VRSNAM* (on page 1059)  
*XAFLAGS* (on page 1061)  
*XAMGR* (on page 1063)  
*XAMGROV* (on page 1066)  
*XIDCNT* (on page 1111)

## NAME

QDDPRMD — DDM Primitive Classes Dictionary

## DESCRIPTION (Semantic)

**Dictionary** QDDSRVDR**Length** \***Class** DICTIONARY

This dictionary contains all DDM classes that describe the primitive data objects used as the descriptive foundation for the DDM architecture and product-unique extensions.

<b>mgrlvln</b>	1
<b>mgrdepls</b>	NIL
dctind	*
objlst	*

## SEE ALSO

**insvar** ACCRDBRM (on page 48)  
 INTTKNRM (on page 448)  
 PKGATHRUL (on page 581)  
 TYPSQLDA (on page 1034)

**mgrdepls** QDDBASD (on page 651)  
 QDDRDBD (on page 657)

**Semantic** ACCDMG (on page 41)  
 ARRAY (on page 101)  
 ASSOCIATION (on page 103)  
 ATTLST (on page 105)  
 BIN (on page 118)  
 BINDR (on page 120)  
 BITDR (on page 123)  
 BITSTRDR (on page 124)  
 BOOLEAN (on page 142)  
 BYTDR (on page 145)  
 BYTSTRDR (on page 146)  
 CHRDR (on page 155)  
 CHRSTRDR (on page 156)  
 CLASS (on page 158)  
 CMDTRG (on page 179)  
 CNSVAL (on page 221)  
 CODPNT (on page 233)  
 CODPNTDR (on page 235)  
 COLLECTION (on page 238)  
 COMMAND (on page 240)  
 CONSTANT (on page 244)  
 DATA (on page 251)  
 DCESEC (on page 252)  
 DDMID (on page 261)  
 DEFINITION (on page 267)  
 DEFLST (on page 268)  
 DFTVAL (on page 276)

*ELMCLS* (on page 330)  
*ENUCLS* (on page 345)  
*ENULEN* (on page 346)  
*ENUVAL* (on page 347)  
*ERROR* (on page 350)  
*EUSRIDDTA* (on page 355)  
*EUSRNPWDDTA* (on page 359)  
*EUSRPWDDTA* (on page 361)  
*EXCSAT* (on page 363)  
*FALSE* (on page 403)  
*FDODSC* (on page 406)  
*FIELD* (on page 415)  
*HELP* (on page 425)  
*HEXDR* (on page 428)  
*HEXSTRDR* (on page 429)  
*IGNORABLE* (on page 435)  
*INFO* (on page 436)  
*INHERITED* (on page 442)  
*INSTANCE\_OF* (on page 444)  
*INTRDBRQS* (on page 445)  
*LENGTH* (on page 472)  
*MAXLEN* (on page 495)  
*MAXVAL* (on page 499)  
*MINLEN* (on page 517)  
*MINLVL* (on page 518)  
*MINVAL* (on page 519)  
*MTLEXC* (on page 524)  
*MTLINC* (on page 525)  
*NAMDR* (on page 526)  
*NAME* (on page 527)  
*NAMSYMDR* (on page 528)  
*NIL* (on page 532)  
*NOTE* (on page 533)  
*NUMBER* (on page 534)  
*OBJECT* (on page 540)  
*OPTIONAL* (on page 568)  
*ORDCOL* (on page 569)  
*OWNER* (on page 577)  
*PLGIN* (on page 609)  
*PRMDMG* (on page 632)  
*QLFATT* (on page 664)  
*RDBINTTKN* (on page 733)  
*REPEATABLE* (on page 758)  
*REQUESTER* (on page 760)  
*REQUIRED* (on page 761)  
*RESERVED* (on page 762)  
*SCALAR* (on page 796)  
*SESDMG* (on page 825)  
*SEVERE* (on page 826)  
*SPCVAL* (on page 831)  
*SPRCLS* (on page 832)  
*STRING* (on page 888)

*SUBSETS* (on page 902)  
*TITLE* (on page 1020)  
*TRUE* (on page 1024)  
*TYPDEF* (on page 1025)  
*TYPDEFNAM* (on page 1027)  
*TYPDEFOVR* (on page 1030)  
*TYPFMLNM* (on page 1033)  
*TYPSQLDA* (on page 1034)  
*USRENCPWD* (on page 1043)  
*USRIDNWPWD* (on page 1045)  
*USRIDONL* (on page 1046)  
*USRIDPWD* (on page 1047)  
*USRSBSPWD* (on page 1049)  
*WARNING* (on page 1060)

## NAME

QDDRDBD — DDM Relational Database Classes Dictionary

## DESCRIPTION (Semantic)

**Dictionary** QDDSRVDR**Length** \***Class** DICTIONARY

The QDDRDBD dictionary contains all DDM classes that describe the commands, replies, and data objects required for relational database functions in DDM.

<b>mgrlvl</b>	3	
<b>mgrdepls</b>	INSTANCE_OF MGRVLVN NOTE	QDDPRMD - DDM Primitive Classes Dictionary 1 The primitives dictionary includes class descriptions for many classes on which QDDRDBD classes depend.
	INSTANCE_OF MGRVLVN NOTE	QDDBASD - DDM Base Classes Dictionary 1 The primitives dictionary includes class descriptions for many classes on which QDDRDBD classes depend.
<b>dctind</b>	*	
<b>objlst</b>	*	

## SEE ALSO

**cmdrpy** *DSCSQLSTT* (on page 304)

**insvar** *ACCRDBRM* (on page 48)  
*DSCSQLSTT* (on page 304)  
*INTTKNRM* (on page 448)

**Semantic** *ABNUOWRM* (on page 39)  
*ACCRDB* (on page 42)  
*ACCRDBRM* (on page 48)  
*ARMCORR* (on page 100)  
*ATMIND* (on page 104)  
*BGNBND* (on page 110)  
*BGNBNDRM* (on page 117)  
*BNDCHKEXS* (on page 125)  
*BNDCHKONL* (on page 126)  
*BNDCRTCTL* (on page 127)  
*BNDERRALW* (on page 128)  
*BNDEXPOPT* (on page 129)  
*BNDEXSOPT* (on page 130)  
*BNDEXSRQR* (on page 131)  
*BNDNERALW* (on page 132)  
*BNDOPT* (on page 133)  
*BNDOPTNM* (on page 134)  
*BNDOPTVL* (on page 135)  
*BNDSQLSTT* (on page 136)

*BNDSTTASM* (on page 141)  
*CLSQRY* (on page 165)  
*CMDSRCID* (on page 178)  
*CMDVLTRM* (on page 181)  
*CMMRQSRM* (on page 182)  
*CMMTYP* (on page 183)  
*CNNTKN* (on page 220)  
*CNTQRY* (on page 222)  
*CODPNTDR* (on page 235)  
*CRRTKN* (on page 246)  
*CSTBITS* (on page 247)  
*CSTMBCS* (on page 248)  
*CSTSBCS* (on page 249)  
*CSTSYSDFT* (on page 250)  
*DECDELCMA* (on page 264)  
*DECDELPRD* (on page 265)  
*DECPRC* (on page 266)  
*DFTDATFMT* (on page 272)  
*DFTPKG* (on page 273)  
*DFTRDBCOL* (on page 274)  
*DFTTIMFMT* (on page 275)  
*DGRIOPRL* (on page 279)  
*DIAGLVL* (on page 284)  
*DMYBLKDATFMT* (on page 288)  
*DMYCMADATFMT* (on page 289)  
*DMYHPNDATFMT* (on page 290)  
*DMYPRDDATFMT* (on page 291)  
*DMYSLHDATFMT* (on page 292)  
*DRPPKG* (on page 293)  
*DSCERRCD* (on page 296)  
*DSCINVRM* (on page 298)  
*DSCRDBTBL* (on page 300)  
*DSCSQLSTT* (on page 304)  
*DTAMCHRM* (on page 320)  
*DUPQRYOK* (on page 322)  
*ENDBND* (on page 336)  
*ENDQRYRM* (on page 342)  
*ENDUOWRM* (on page 343)  
*EURDATFMT* (on page 353)  
*EURTIMFMT* (on page 354)  
*EXCSAT* (on page 363)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSET* (on page 377)  
*EXCSQLSTT* (on page 381)  
*EXPALL* (on page 393)  
*EXPNON* (on page 394)  
*EXPREOPT* (on page 395)  
*EXPYES* (on page 396)  
*EXTDTA* (on page 397)  
*EXTDTAOVR* (on page 399)  
*FDODSCOFF* (on page 407)  
*FDODTAOFF* (on page 409)



*FDOPRMOFF* (on page 413)  
*FDOTRPOFF* (on page 414)  
*FIXROWPRC* (on page 417)  
*FRCFIXROW* (on page 424)  
*HMSBLKTIMFMT* (on page 431)  
*HMSCLNTIMFMT* (on page 432)  
*HMSCMATIMFMT* (on page 433)  
*HMSPRDTIMFMT* (on page 434)  
*INTRDBRQS* (on page 445)  
*INTTKNRM* (on page 448)  
*ISODATFMT* (on page 451)  
*ISOLVLALL* (on page 452)  
*ISOLVLCHG* (on page 453)  
*ISOLVLC* (on page 454)  
*ISOLVLNC* (on page 455)  
*ISOLVLR* (on page 456)  
*ISOTIMFMT* (on page 457)  
*JISDATFMT* (on page 458)  
*JISTIMFMT* (on page 459)  
*JULBLKDATFMT* (on page 460)  
*JULCMADATFMT* (on page 461)  
*JULHPNDATFMT* (on page 462)  
*JULPRDDATFMT* (on page 463)  
*JULSLHDATFMT* (on page 464)  
*LMTBLKPRC* (on page 475)  
*LOCDATFMT* (on page 482)  
*LOCTIMFMT* (on page 483)  
*MAXBLKEXT* (on page 494)  
*MAXRSLCNT* (on page 497)  
*MAXSCTNBR* (on page 498)  
*MDYBLKDATFMT* (on page 500)  
*MDYCMADATFMT* (on page 501)  
*MDYHPNDATFMT* (on page 502)  
*MDYPRDDATFMT* (on page 503)  
*MDYSLHDATFMT* (on page 504)  
*NBRROW* (on page 530)  
*OPNQFLRM* (on page 554)  
*OPNQRY* (on page 555)  
*OPNQRYRM* (on page 566)  
*OUTEXP* (on page 571)  
*OUTOVR* (on page 572)  
*OUTOVRANY* (on page 574)  
*OUTOVRFRS* (on page 575)  
*OUTOVROPT* (on page 576)  
*PKGATHKP* (on page 579)  
*PKGATHOPT* (on page 580)  
*PKGATHRUL* (on page 581)  
*PKGATHRVK* (on page 584)  
*PKGBNARM* (on page 585)  
*PKGBPARM* (on page 586)  
*PKGCNSTKN* (on page 588)  
*PKGDFTCST* (on page 590)

*PKGID* (on page 591)  
*PKGISOLVL* (on page 592)  
*PKGNAME* (on page 594)  
*PKGNAMECSN* (on page 596)  
*PKGNAMECT* (on page 598)  
*PKGOWNID* (on page 600)  
*PKGRPLALW* (on page 602)  
*PKGRPLNA* (on page 603)  
*PKGRPLOPT* (on page 604)  
*PKGSN* (on page 606)  
*PKGSNLST* (on page 607)  
*PRCCNVCD* (on page 621)  
*PRDDTA* (on page 630)  
*PRPSQLSTT* (on page 636)  
*QRYATTSET* (on page 666)  
*QRYBLK* (on page 670)  
*QRYBLKCTL* (on page 672)  
*QRYBLKEXA* (on page 674)  
*QRYBLKFCT* (on page 675)  
*QRYBLKFLX* (on page 676)  
*QRYBLKSZ* (on page 678)  
*QRYBLKTYP* (on page 679)  
*QRYCLSIMP* (on page 680)  
*QRYCLSRLS* (on page 682)  
*QRYDSC* (on page 685)  
*QRYDTA* (on page 686)  
*QRYINSID* (on page 688)  
*QRYNOPRM* (on page 690)  
*QRYPOPRM* (on page 691)  
*QRYPRCTYP* (on page 692)  
*QRYROWNBR* (on page 694)  
*QRYROWSET* (on page 697)  
*QRYSCRCUR* (on page 706)  
*QRYSCRFST* (on page 707)  
*QRYSCRLST* (on page 708)  
*QRYSCRNXT* (on page 709)  
*QRYSCRPRI* (on page 712)  
*RDB* (on page 718)  
*RDBACCCL* (on page 723)  
*RDBACCRM* (on page 724)  
*RDBAFLRM* (on page 725)  
*RDBALWUPD* (on page 726)  
*RDBATHRM* (on page 727)  
*RDBCMM* (on page 728)  
*RBCOLID* (on page 732)  
*RDBINTTKN* (on page 733)  
*RDBNACRM* (on page 734)  
*RDBNAM* (on page 736)  
*RDBNFNRM* (on page 739)  
*RDBRLLBCK* (on page 745)  
*RDBRLSCMM* (on page 748)  
*RDBRLSCNV* (on page 749)

*RDBRLSOPT* (on page 750)  
*RDBUPDRM* (on page 751)  
*REBIND* (on page 753)  
*RSLSETFLG* (on page 783)  
*RTNEXTALL* (on page 791)  
*RTNEXTDTA* (on page 792)  
*RTNEXTROW* (on page 793)  
*RSLSETRM* (on page 785)  
*RTNSETSTT* (on page 794)  
*RTNSQLDA* (on page 795)  
*SQL* (on page 835)  
*SQLATTR* (on page 854)  
*SQLAM* (on page 847)  
*SQLCARD* (on page 855)  
*SQLCINRD* (on page 857)  
*SQLCSRHLD* (on page 858)  
*SQLDARD* (on page 859)  
*SQLDTA* (on page 860)  
*SQLDTARD* (on page 862)  
*SQLERRRM* (on page 864)  
*SQLOBNAM* (on page 866)  
*SQLRSLRD* (on page 867)  
*SQLSTT* (on page 868)  
*SQLSTTNBR* (on page 870)  
*SQLSTTVRB* (on page 871)  
*SRVLCNT* (on page 875)  
*SRVLSRV* (on page 876)  
*SRVLST* (on page 878)  
*SRVPRTY* (on page 883)  
*STRDELAP* (on page 886)  
*STRDELQ* (on page 887)  
*STTASMEUI* (on page 893)  
*STTDATFMT* (on page 894)  
*STTDECDEL* (on page 897)  
*STTSCCLS* (on page 898)  
*STTSTRDEL* (on page 899)  
*STTTIMFMT* (on page 900)  
*SUBSETS* (on page 902)  
*TYPSQLDA* (on page 1034)  
*UOWDSP* (on page 1037)  
*USADATFMT* (on page 1041)  
*USATIMFMT* (on page 1042)  
*YMDBLKDATFMT* (on page 1113)  
*YMDCMADATFMT* (on page 1114)  
*YMDHPNDATFMT* (on page 1115)  
*YMDPRDDATFMT* (on page 1116)  
*YMDSLHDATFMT* (on page 1117)

## NAME

QDDTTRD — DDM Tutorial Objects Dictionary

## DESCRIPTION (Semantic)

**Dictionary** QDDSRVDR**Length** \***Class** DICTIONARY

This dictionary contains help objects that provide tutorial information about DDM.

---

objlst	*
dctind	*

---

## SEE ALSO

**Semantic**

- AGNCMDPR* (on page 64)
- AGNRPYPR* (on page 67)
- APPCMNFL* (on page 68)
- APPCMNI* (on page 72)
- APPCMNT* (on page 76)
- APPSRCCD* (on page 79)
- APPSRCCR* (on page 86)
- APPSRCER* (on page 91)
- APPTRGER* (on page 95)
- CMNLYR* (on page 194)
- CMNOVR* (on page 201)
- CONCEPTS* (on page 243)
- DCESECOVR* (on page 253)
- DCESECTKN* (on page 257)
- DDM* (on page 260)
- DDMOBJ* (on page 262)
- DHENC* (on page 280)
- DSS* (on page 308)
- DTAOVR* (on page 321)
- EDTASECOVR* (on page 323)
- EXTENSIONS* (on page 400)
- FDOCA* (on page 404)
- INHERITANCE* (on page 437)
- LVLCMP* (on page 486)
- MGROVR* (on page 514)
- NWPWDSEC* (on page 535)
- OBJOVR* (on page 544)
- OOPOVR* (on page 547)
- OSFDCE* (on page 570)
- PLGINOVR* (on page 615)
- PRCOVR* (on page 628)
- PWDENC* (on page 644)
- PWDSBS* (on page 646)
- PWDSEC* (on page 649)
- RDBOVR* (on page 740)
- RESYNOVR* (on page 764)
- SECOVR* (on page 819)

*SNASECOVR* (on page 828)  
*SRVOVR* (on page 881)  
*STRLYR* (on page 890)  
*SUBSETS* (on page 902)  
*SYNCMNBK* (on page 924)  
*SYNCMNCM* (on page 926)  
*SYNCMNFL* (on page 928)  
*SYNCMNI* (on page 931)  
*SYNCMNT* (on page 935)  
*SYNCPTOV* (on page 944)  
*TASK* (on page 991)  
*TCPCMNFL* (on page 992)  
*TCPCMNI* (on page 994)  
*TCPCMNT* (on page 997)  
*TCPIPOVR* (on page 1000)  
*TCPSRCCD* (on page 1007)  
*TCPSRCCR* (on page 1010)  
*TCPSRCER* (on page 1013)  
*TCPTRGER* (on page 1015)  
*USRIDSEC* (on page 1048)  
*USRSECOVR* (on page 1050)

**NAME**

QLFATT — Qualified Attribute

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'0007'

**Length** \*

**Class** CLASS

**Sprcls** ASSOCIATION - Name with Value Association

Qualified Attribute (QLFATT) Association specifies the qualified attributes by listing subattributes. The key variable of the QLFATT object contains the attribute being qualified—the key attribute. The value variable of the QLFATT object contains an attribute list (ATTLST) that qualifies the meaning or use of the key attribute.

When specified as an attribute in an ATTLST, the key attribute of the QLFATT replaces the QLFATT. Thus, QLFATT objects can be specified within ATTLST objects to any required level of nesting.

In this specification, qualified attributes are formatted as attributes with an indented list of subattributes following them. For example,

```
QLFATT ( ENUVAL ( 57 )
ATTLST
          NOTE ( 'This value reverses all commands.' ) ) )
```

is printed as:

```
          ENUVAL    57
MINLVL
          NOTE      This value reverses all commands.
```

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'0007'	
<b>key</b>	SPRCLS REQUIRED	OBJECT - Self-identifying Data
<b>value</b>	INSTANCE_OF REQUIRED	ATTLST - Attribute List
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

None.

**NAME**

QRYATTSCR — Query Attribute for Scrollability

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2149'  
**Length** \*  
**Class** CLASS  
**Sprcls** BOOLEAN - Logical Value

Query Attribute for Scrollability (QRYATTSCR) indicates whether a cursor is scrollable or non-scrollable.

QRYATTSCR is returned on the OPNQRYM.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	5	
class	X'2149'	
value	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	LENGTH	2
	ENUVAL	X'F1' - TRUE - True State
	NOTE	The cursor is scrollable.
	ENUVAL	X'F0' - FALSE - False State
	NOTE	The cursor is non-scrollable.
	DFTVAL	X'F0' - FALSE - False State
	MINLVL	7
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** OPNQRYM (on page 566)  
**Semantic** CNTQRY (on page 222)  
 LMTBLKPRC (on page 475)

**NAME**

QRYATTSET — Query Attribute for Rowset

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'214A'  
**Length** \*  
**Class** CLASS  
**Sprcls** BOOLEAN - Logical Value

Query Attribute for Rowset (QRYATTSET) indicates whether a cursor is enabled for rowset positioning. It specifies that the cursor or result set can be used to return either a single row or multiple rows, as a rowset, with a single FETCH statement. Cursors declared WITH ROWSET POSITIONING can be used with row-positioned FETCH statements.

QRYATTSET is returned on the OPNQRYM.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	5	
class	X'214A'	
value	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	LENGTH	2
	ENUVAL	X'F1' - TRUE - True State
	NOTE	The cursor is enabled for rowset positioning.
	ENUVAL	X'F0' - FALSE - False State
	NOTE	The cursor is not enabled for rowset positioning.
	DFTVAL	X'F0' - FALSE - False State
	MINLVL	7
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** CNTQRY (on page 222)  
**Semantic** CNTQRY (on page 222)



**NAME**

QRYATTSNS — Query Attribute for Sensitivity

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2157'

**Length** \*

**Class** CLASS

**Sprcls** BIN - Binary Integer Number

Query Attribute for Sensitivity (QRYATTSNS) indicates the sensitivity of an opened cursor to changes made to the underlying base table.

QRYATTSNS is returned on the OPNQRYM.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	5	
class	X'2157'	
value	INSTANCE_OF	BIN - Binary Number Field
	LENGTH	8
	ENUVAL	0 QRYUNK
	NOTE	This query attribute is unknown or undefined for this cursor.
	ENUVAL	1 QRYINS
	NOTE	The cursor is <i>insensitive</i> to changes made to rows underlying the result table after it has been materialized.
	ENUVAL	2 QRYSNSSTC
	NOTE	The cursor is <i>sensitive</i> to changes made to rows underlying the result table after it has been materialized and the result table has <i>static</i> size and ordering.
	ENUVAL	3 QRYSNSDYN
	NOTE	The cursor is <i>sensitive</i> to changes made to rows underlying the result table after it has been materialized and the result table has <i>dynamic</i> size and ordering.
	DFTVAL	0 QRYUNK
	MINLVL	7
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

## SEE ALSO

<b>insvar</b>	<i>OPNQRYRM</i> (on page 566)
<b>Semantic</b>	<i>FIXROWPRC</i> (on page 417)
	<i>LMTBLKPRC</i> (on page 475)

**NAME**

QRYATTUPD — Query Attribute for Updatability

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2150'  
**Length** \*  
**Class** CLASS  
**Sprcls** Binary Integer Number

Query Attribute for Updatability (QRYATTUPD) indicates the updatability of an opened cursor. QRYATTUPD is returned on the OPNQRYRM.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	5	
class	X'2150'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	8
	ENUVAL	0 QRYUNK
	NOTE	This query attribute is unknown or undefined for this cursor.
	ENUVAL	1 QRYRDO
	NOTE	The cursor is <i>read-only</i> .
	ENUVAL	2 QRYDEL
	NOTE	The cursor allows <i>read</i> and <i>delete</i> operations.
	ENUVAL	4 QRYUPD
	NOTE	The cursor allows <i>read</i> , <i>delete</i> , and <i>update</i> operations.
	DFTVAL	0 QRYUNK
	MINLVL	7
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** OPNQRYRM (on page 566)

## NAME

QRYBLK — Query Block

## DESCRIPTION (Semantic)

**Dictionary** QDDTTRD**Length** \***Class** HELP**Sprcls** Query Block

Query Block (QRYBLK) is the unit of transmission for query or result set data and serves as the basis for a method to control how much query data is returned with each OPNQRY, EXCSQLSTT, or CNTQRY command when the LMTBLKPRC is in effect or if the *qryrowset* parameter was specified on the OPNQRY, EXCSQLSTT, or CNTQRY command.

Each query block is a QRYDTA reply data object that contains row data from one or more answer set rows. The size of each query block is determined by the *qryblksz* parameter on the command. Query blocks have the following characteristics, according to their type. The server selects the type of query blocks to return based on its capabilities and preferences and uses the *qryblktyp* parameter on the OPNQRYRM to indicate its choice.

- **Exact blocking:** The *qryblksz* is an exact value. 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 of the query block 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. Each query block in this case is called an exact query block.
- **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 all 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. Each query block in this case is called a flexible query block.

The following also applies to query blocks (see Blocking Rules BF, BS, CH, and QP in the DRDA Reference for detailed information):

- If answer set data is returned in reply to an OPNQRY, EXCSQLSTT, or CNTQRY command, then there is at least one query block returned and the query block(s) returned must contain the end of at least one complete row.
- The *qryblksz* parameter can be changed on any CNTQRY command. In addition to using the *qryblksz* parameter to control the amount of data it receives, the source system can also use the *maxblkext* parameter to control how many query blocks are to be returned. The target system cannot return more than that number of extra query blocks, but may return fewer.

See *CNTQRY* (on page 222), *EXCSQLSTT* (on page 381), *FIXROWPRC* (on page 417), *LMTBLKPRC* (on page 475), *MAXBLKEXT* (on page 494), *OPNQRY* (on page 555), and *OPNQRY* (on page 555).

**SEE ALSO**

**Semantic**      *EXTDTAOVR* (on page 399)  
*FIXROWPRC* (on page 417)  
*LMTBLKPRC* (on page 475)  
*OPNQRY* (on page 555)  
*QRYBLKEXA* (on page 674)  
*QRYBLKFLX* (on page 676)  
*QRYBLKSZ* (on page 678)  
*QRYBLKTYP* (on page 679)

NAME

QRYBLKCTL — Query Block Protocol Control

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'2132'  
**Length** \*  
**Class** CLASS  
**Sprcls** STRING - String

Query Block Protocol Control (QRYBLKCTL) String controls the type of query block protocol used when a query is opened. When the parameter is specified in the OPNQRY command, it controls the query protocol used for the specific query being opened. When the parameter is specified in the BGNBND command, it controls the query protocols all queries in the package use unless the OPNQRY command overrides it.

A database cursor is ambiguous if it is not declared for FETCH ONLY or UPDATE, not the target of a WHERE\_CURRENT\_OF clause on an SQL UPDATE or DELETE statement, and if the package has dynamic SQL capability. All other database cursors are certain.

When the database cursor is certain and the *qryblkctl* parameter takes the value FIXROWPRC or LMTBLKPRC, then the protocols used are defined in the terms FIXROWPRC and LMTBLKPRC.

The value FRCFIXROW means that the target SQLAM must use the fixed row query protocol for all database cursors (see *FRCFIXROW* (on page 424)).

The value FIXROWPRC means that the target SQLAM must use the fixed row query protocol for all queries that have ambiguous database cursors.

The value LMTBLKPRC means that the target SQLAM must use the limited block query protocol for all queries that have ambiguous database cursors, except if the cursor requires FIXROWPRC because of the attributes of the cursor. See *FIXROWPRC* (on page 417) for details.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	6	
class	X'2132'	
value	INSTANCE_OF	CODPNTDR - Codepoint Data Representation
	ENUVAL	X'2418' - FIXROWPRC - Fixed Row Query Protocol
	ENUVAL	X'2417' - LMTBLKPRC - Limited Block Query Protocol
	ENUVAL	X'2410' - FRCFIXROW - Force Fixed Row Query Protocol
	DFTVAL	X'2418' - FIXROWPRC - Fixed Row Query Protocol
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

- |                 |   |
|-----------------|---|
| <b>insvar</b>   | <i>BGNBND</i> (on page 110)<br><i>OPNQRY</i> (on page 555)  |
| <b>Semantic</b> | <i>ENDBND</i> (on page 336)<br><i>FIXROWPRC</i> (on page 417)<br><i>LMTBLKPRC</i> (on page 475)<br><i>OPNQRY</i> (on page 555)<br><i>REBIND</i> (on page 753) |

**NAME**

QRYBLKEXA — Query Block Exact Type

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Length** \*

**Class** CONSTANT — Constant Value

On an OPNQRYRM, the QRYBLKEXA value in the QRYBLKTYP parameter indicates that the query blocks returned for this cursor will be exact blocks. See *QRYBLK* (on page 670) for a definition of query block and a definition of the types of query blocks.

---

value 0

**SEE ALSO**

**insvar** *QRYBLK* (on page 670)  
*QRYBLKTYP* (on page 679)

**Semantic** *CNTQRY* (on page 222)



**NAME**

QRYBLKFACT — Query Blocking Factor

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'215F'  
**Length** \*  
**Class** CLASS  
**Sprcls** BIN - Binary Integer Number

Query Blocking Factor (QRYBLKFACT) contains the value of the blocking factor which is a limit imposed by the target server in terms of the number of rows that can be blocked at a time for a query. This limit is not influenced by the source server via any parameter that may or may not have been explicitly specified on the EXCSQLSTT or OPNQRY command. The blocking factor may be set via a configuration setting on the target server, or it may be a result of a clause on the SQL statement that is associated with the query. This value 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.

This instance variable is returned on the OPNQRYRM reply message for a cursor that is opened via the EXCSQLSTT or OPNQRY command.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	5	
<b>class</b>	X'215F'	
<b>value</b>	INSTANCE_OF	BINDR - Binary Number Field
	MINLVL	7
	LENGTH	32
	MINVAL	1
	SPCVAL	0
	NOTE	A value of 0 indicates there is no applicable blocking limit which is self-imposed by the target server without influence from the source server.
	DFTVAL	0
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** BINDR (on page 120)  
 OPNQRYRM (on page 566)  
**Semantic** CLASS (on page 158)  
 EXCSQLSTT (on page 381)  
 OPNQRY (on page 555)  
 OPNQRYRM (on page 566)  
 QDDRDBD (on page 657)

**NAME**

QRYBLKFLX — Query Block Flexible Type

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Length** \*

**Class** CONSTANT — Constant Value

On an OPNQRYRM, the QRYBLKFLX value in the QRYBLKTYP parameter indicates that the query blocks returned for this cursor will be flexible blocks. See *QRYBLK* (on page 670) for a definition of query block and a definition of the types of query blocks.

---

value 1

**SEE ALSO**

**insvar** *QRYBLK* (on page 670)  
*QRYBLKTYP* (on page 679)

**NAME**

QRYBLKRST — Query Block Reset

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2154'  
**Length** \*  
**Class** CLASS  
**Sprcls** BOOLEAN - Logical Value

Query Block Reset (QRYBLKRST) on a CNTQRY command indicates whether any pending partial row and pending query blocks (QRYDTA and EXTDTA) are to be discarded.

This operation is used in conjunction with scrollable, non-rowset cursors to terminate an incomplete DRDA rowset.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	5	
class	X'2154'	
value	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	LENGTH	2
	ENUVAL	X'F1' - TRUE - True State
	NOTE	Pending query blocks are to be reset.
	ENUVAL	X'F0' - FALSE - False State
	NOTE	Pending query block are not to be reset.
	DFTVAL	X'F0' - FALSE - False State
	MINLVL	7
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** CNTQRY (on page 222)  
**Semantic** QRYROWSET (on page 697)

**NAME**

QRYBLKSZ — Query Block Size

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2114'  
**Length** \*  
**Class** CLASS  
**Sprcls** BIN - Binary Integer Number

Query Block Size (QRYBLKSZ) controls the size of each query block that the requester wants the server to use when returning answer set data in query blocks. How this parameter is used in determining the size of the query block depends on what type of query block the server will return. See *QRYBLK* (on page 670) for a definition of query block, a definition of the types of query blocks, and how the size of the query block is determined.

The query block size for the target SQLAM must always conform with the query block size specified on *OPNQRY*, *CNTQRY*, or *EXCSQLSTT* containing an SQL statement that invokes an external procedure.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	8	
<b>class</b>	X'2114'	
<b>value</b>	INSTANCE_OF	UNSBINDR - Unsigned Binary Data Representation
	LENGTH	32
	MINVAL	512
	MAXVAL	10,485,760 (10M)
	MINLVL	7
	REQUIRED	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *CNTQRY* (on page 222)  
*EXCSQLSTT* (on page 381)  
*OPNQRY* (on page 555)  
**Semantic** *EXCSQLSTT* (on page 381)  
*QRYBLK* (on page 670)  
*QRYBLKTYP* (on page 679)

**NAME**

QRYBLKTYP — Query Block Type

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2133'  
**Length** \*  
**Class** CLASS  
**Title** BIN - Binary Integer Number

Query Block Type (QRYBLKTYP) specifies the type of query blocks the target SQLAM will use to return answer set data for a query. The primary characteristic which differentiates each type is the size of the query block, as controlled by the QRYBLKSZ parameter that is specified with the command. See *QRYBLK* (on page 670) for a definition of query block and a definition of the types of query blocks.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
<b>length</b>	5	
<b>class</b>	X'2133'	
<b>value</b>	INSTANCE_OF	BINDR - Binary Number Field
	ENUVAL	0 QRYBLKEXA
	NOTE	Every query block is exactly the size specified in the QRYBLKSZ parameter, except for possibly the last query block which may be shorter.
	ENUVAL	1 QRYBLKFLX
	NOTE	Every query block is at least the size specified in the QRYBLKSZ parameter, except for possibly the last query block which may be shorter.
	DFTVAL	0 QRYBLKEXA
	MINLVL	7
	OPTIONAL	
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** OPNQRYRM (on page 566)  
**Semantic** CNTQRY (on page 222)  
 OPNQRY (on page 555)  
 QRYBLKEXA (on page 674)  
 QRYBLKFLX (on page 676)

NAME

QRYCLSIMP — Query Close Implicit

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'215D'  
**Length** \*  
**Class** CLASS  
**Sprcls** STRING - String

Query Close Implicit (QRYCLSIMP) controls whether the target server implicitly closes a non-scrollable query upon end of data (SQLSTATE 02000).

A query is considered to be closed implicitly by the target server if a Close Query (CLSQRV) command has not been received from the source server when the cursor gets closed.

If the query is scrollable, the target server does not implicitly close the query regardless of the QRYCLSIMP value specified which is ignored. Otherwise, the QRYCLSIMP value results in one of the following behaviors:

- The target server determines whether to implicitly close the query or not upon SQLSTATE 02000 based on the cursor type. For instance, the target server may choose to implicitly close all non-scrollable cursors except those declared with the HOLD attribute.
- The target server must implicitly close the query upon SQLSTATE 02000.
- The target server must not implicitly close the query upon SQLSTATE 02000. Note that the target server is still allowed to implicitly close the query under other query terminating (error) conditions.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	5	
<b>class</b>	X'215D'	
<b>value</b>	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	MINLVL	7
	LENGTH	2
	ENUVAL	X'00'
	NOTE	Target server determines whether to implicitly close the cursor or not upon SQLSTATE 02000 based on the cursor type.
	ENUVAL	X'01'
	NOTE	Target server must implicitly close the cursor upon SQLSTATE 02000.
	ENUVAL	X'02'
	NOTE	Target server must not implicitly close the cursor upon SQLSTATE 02000.
	IGNORABLE	If query is scrollable.

	DFTVAL	X'00'
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

<b>insvar</b>	<i>HEXSTRDR</i> (on page 429) <i>OPNQRY</i> (on page 555)
<b>Semantic</b>	<i>CLASS</i> (on page 158) <i>CLSQRY</i> (on page 165) <i>CNTQRY</i> (on page 222) <i>FIXROWPRC</i> (on page 417) <i>LMTBLKPRC</i> (on page 475) <i>QDDRDBD</i> (on page 657) <i>QRYCLSIMP</i> (on page 680)

**NAME**

QRYCLSRLS — Query Close Lock Release

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'215E'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Query Close Lock Release (QRYCLSRLS) controls whether read locks held for a cursor are released when the query is closed. This is regardless of whether the query is closed implicitly by the target server (as a result of SQLSTATE 02000 or otherwise a query terminating error condition) or explicitly as requested by the source server.

Note that even when requested to do so, the target server may not necessarily be able to release all read locks held by a cursor since they may be held for other operations or activities. Failure to release all read locks for a cursor does not constitute an error condition in this case.

QRYCLSRLS can be specified by a source server on the OPNQRY command when a query is opened.

QRYCLSRLS can also be specified by a source server on the CLSQRY command when the query is explicitly closed. When specified in this manner, the QRYCLSRLS value overrides any setting which may have been specified or defaulted to previously on the corresponding OPNQRY command.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	5	
class	X'215E'	
value	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	MINLVL	7
	LENGTH	2
	ENUVAL	X'00'
	NOTE	Do not release read locks when the query is closed.
	ENUVAL	X'01'
	NOTE	Release read locks when the query is closed.
	DFTVAL	X'00'
	NOTE	For an OPNQRY command.
	NOTE	For a CLSQRY command, the default is the value specified or defaulted to previously on the corresponding OPNQRY command.
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	



**SEE ALSO**

- insvar**            *CLSQRY* (on page 165)  
                      *HEXSTRDR* (on page 429)  
                      *OPNQRY* (on page 555)
  
- Semantic**        *CLASS* (on page 158)  
                      *CLSQRY* (on page 165)  
                      *OPNQRY* (on page 555)  
                      *QDDRDBD* (on page 657)

**NAME**

QRYDEL — Query Deletable

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Length** \*

**Class** CONSTANT

On an OPNQRYRM, the QRYDEL value in the QRYATTUPD parameter indicates that the query being opened allows read and delete operations on the cursor.

---

value	2
-------	---

**SEE ALSO**

**Semantic**      *EXCSQLSTT* (on page 381)  
                   *OPNQRY* (on page 555)  
                   *OPNQRYRM* (on page 566)  
                   *QRYATTUPD* (on page 669)

**NAME**

QRYDSC — Query Answer Set Description

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'241A'  
**Length** \*  
**Class** CLASS  
**Sprcls** FDODSC - FD:OCA Data Descriptor

Query Answer Set Description (QRYDSC) specifies the information about the answer set data that is returned from a query or result set. Refer to the Blocking rules (BF, BS, and CH) in the DRDA Reference for an explanation of the FD:OCA description for the data value of this object and for detailed information about how this object is formatted.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
<b>length</b>	*
<b>class</b>	X'241A'
<b>value</b>	INSTANCE_OF BYTSTRDR - Byte String REQUIRED
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO**

**cmdtda** CNTQRY (on page 222)  
**rpydta** EXCSQLSTT (on page 381)  
 OPNQRY (on page 555)  
**Semantic** CNTQRY (on page 222)  
 EXCSQLSTT (on page 381)  
 EXTDTAOVR (on page 399)  
 FIXROWPRC (on page 417)  
 LMTBLKPRC (on page 475)  
 OPNQRY (on page 555)  
 OUTOVR (on page 572)  
 OUTOVRANY (on page 574)  
 OUTOVRFRS (on page 575)  
 QRYBLK (on page 670)  
 QRYDTA (on page 686)

NAME

QRYDTA — Query Answer Set Data

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'241B'  
**Length** \*  
**Class** CLASS  
**Sprcls** FDOTA - FD:OCA Data

Query Answer Set Data (QRYDTA) contains some or all of the answer set data resulting from a query. The QRYDSC that the OPNQRY or EXCSQLSTT that invoked a stored procedure returned at the beginning of the query object describes the contents of the QRYDTA.

A QRYDTA object can optionally be generated in a streamed manner (see Layer B Streaming in DSS (on page 308) for details).

With certain SECMECs, the DSS carrier containing the QRYDTA FD:OCA object must be encrypted. See EDTASECOVR (on page 323) for details.)

<b>clsvar</b>	NIL
<b>insvar</b>	CLASS INSTANCE VARIABLES
<b>length</b>	*
<b>class</b>	X'241B'
<b>value</b>	INSTANCE_OF BYTSTRDR - Byte String REQUIRED
<b>clscmd</b>	NIL
<b>inscmd</b>	NIL

SEE ALSO

**insvar** RTNEXTDTA (on page 792)  
**rpydta** CNTQRY (on page 222)  
 EXCSQLSTT (on page 381)  
 OPNQRY (on page 555)  
**Semantic** CNTQRY (on page 222)  
 DHENC (on page 280)  
 EDTASECOVR (on page 323)  
 EUSRIDDTA (on page 355)  
 EUSRPWDDTA (on page 361)  
 EXTDTA (on page 397)  
 EXTDTAOVR (on page 399)  
 EXCSQLSTT (on page 381)  
 FIXROWPRC (on page 417)  
 LMTBLKPRC (on page 475)  
 OPNQRY (on page 555)  
 QRYBLK (on page 670)  
 RTNEXTALL (on page 791)  
 RTNEXTROW (on page 793)

**NAME**

QRYINS — Query Insensitive to Changes

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Length** \***Class** CONSTANT

On an OPNQRYRM, the QRYINS value in the QRYATTSNS parameter indicates that the query being opened is not sensitive to inserts, updates, or delete operations made to the underlying rows of the result table.

Insensitive scrollable cursors are read-only.

---

 value

1

**SEE ALSO**

**Semantic**      *EXCSQLSTT* (on page 381)  
                   *LMTBLKPRC* (on page 475)  
                   *OPNQRY* (on page 555)  
                   *OPNQRYRM* (on page 566)  
                   *QRYATTUPD* (on page 669)

**NAME**

QRYINSID — Query Instance Identifier

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'215B'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

Query Instance Identifier (QRYINSID) uniquely identifies the instance of a query. Its contents are implementation-specific and are unarchitected by DDM (see Note 1).

The QRYINSID parameter is returned on the OPNQRYRM reply message by the target server when a query is opened. The source server, when sending out any command (for example, CNTQRY and CLSQRY) that subsequently references this query, must then provide the QRYINSID value in order to uniquely identify the instance of the query.

**Note 1**

The following is an illustration of how the QRYINSID may be implemented by a target server which merely serves as an example.

The QRYINSID value can be a concatenation of the Query Nesting Level and the Query Unique Identifier.

The Query Nesting Level identifies the level at which a query is opened. For example, a target server may return a Query Nesting Level of 0 for a query that is opened by an application executing at a source server. If this application then executes a stored procedure which opens a query, the target server can assign a Query Nesting Level of 1 for this query. And if this stored procedure then calls itself recursively, the query opened at that level may be assigned a Query Nesting Level of 2, and so on.

The Query Unique Identifier provides a sequence number for a query. For example, a target server may return a Query Unique Identifier of 0 for a query that is opened from within a stored procedure. If this query is still active when the stored procedure is exited, and the stored procedure is executed again, the target server can assign a Query Unique Identifier of 1 for the query opened by this subsequent stored procedure invocation.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	12	
class	X'215C'	
value	INSTANCE_OF LENGTH	BINDR - Binary Number Field 64
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

<b>insvar</b>	<i>BINDR</i> (on page 120) <i>CLSQRY</i> (on page 165) <i>CNTQRY</i> (on page 222) <i>DSCSQLSTT</i> (on page 304) <i>EXCSQLIMM</i> (on page 371) <i>EXCSQLSTT</i> (on page 381) <i>OPNQRYRM</i> (on page 566)
<b>Semantic</b>	<i>CLASS</i> (on page 158) <i>CLSQRY</i> (on page 165) <i>CNTQRY</i> (on page 222) <i>OPNQRYRM</i> (on page 566) <i>QDDRDBD</i> (on page 657) <i>SCALAR</i> (on page 796)

**NAME**

QRYNOPRM — Query Not Open

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2202'  
**Length** \*  
**Class** CLASS  
**Sprcls** RPYMSG - Reply Message

Query Not Open (QRYNOPRM) Reply Message is issued if a CNTQRY or CLSQRY command is issued for a query that is not open. A previous ENDQRYRM, ENDUOWRM, or ABNUOWRM reply message might have terminated the command.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'2202'	
<b>pkgnamcsn</b>	INSTANCE_OF REQUIRED	PKGNAMECSN - RDB Package Name, Consistency Token, and Section Number
<b>rdbnam</b>	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF ENUVAL ENUVAL REQUIRED	SVRCOD - Severity Code 4 - WARNING - Warning Severity Code 8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**cmdrpy** CLSQRY (on page 165)  
CNTQRY (on page 222)

**Semantic** CLSQRY (on page 165)  
CNTQRY (on page 222)  
FIXROWPRC (on page 417)  
LMTBLKPRC (on page 475)



**NAME**

QRYPOPRM — Query Previously Opened

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'220F'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

Query Previously Opened (QRYPOPRM) Reply Message is issued when an OPNQRY command is issued for a query that is already open. A previous OPNQRY command might have opened the query which may not be closed.

**DSS Carrier:** RPYDSS

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'220F'	
pkgnamcsn	INSTANCE_OF REQUIRED	PKGNAMECSN - RDB Package Name, Consistency Token, and Section Number
rdbnam	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
svrcod	INSTANCE_OF REQUIRED	SVRCOD - Severity Code
	ENUVAL	8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**cmdrpy** OPNQRY (on page 555)

**Semantic** CNTQRY (on page 222)  
 DUPQRYOK (on page 322)  
 FIXROWPRC (on page 417)  
 LMTBLKPRC (on page 475)  
 OPNQRY (on page 555)  
 QRYBLK (on page 670)

**NAME**

QRYPRCTYP — Query Protocol Type

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2102'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Query Protocol Type (QRYPRCTYP) String specifies the type of query protocol the target SQLAM uses.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	6	
class	X'2102'	
value	INSTANCE_OF	CODPNTDR - Codepoint Data Representation
	ENUVAL	X'2418' - FIXROWPRC - Fixed Row Query Protocol
	ENUVAL	X'2417' - LMTBLKPRC - Limited Block Query Protocol
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** CNTQRY (on page 222)  
OPNQRYSM (on page 566)

**Semantic** QRYPRCTYP  
QRYROWSET (on page 697)

**NAME**

QRYRDO — Query Read-only

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Length** \***Class** CONSTANT

On an OPNQRYRM, the QRYRDO value in the QRYATTUPD parameter indicates that the query being opened is a read-only cursor.

---

**value**

1

**SEE ALSO**

**Semantic**      *EXCSQLSTT* (on page 381)  
                  *OPNQRY* (on page 555)  
                  *OPNQRYRM* (on page 566)  
                  *QRYATTUPD* (on page 669)

**NAME**

QRYROWNBR — Query Row Number

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'213D'  
**Length** \*  
**Class** CLASS  
**Sprcls** BIN - Binary Integer Number

Query Row Number (QRYROWNBR) specifies a row number when using scrollable cursors.

The meaning of QRYROWNBR depends on whether the cursor is a rowset cursor or a non-rowset cursor, and upon the value of QRYROWSET. If QRYROWSET is specified, then a rowset operation is being performed. If QRYROWSET is not specified, then a single row operation is being performed.

See the DRDA Reference, Appendix B for a discussion of the different cursor types.

For non-rowset cursors with no value specified for QRYROWSET, QRYROWNBR indicates that the cursor should be positioned on a single row. For non-rowset cursors with a value specified for QRYROWSET, QRYROWNBR indicates that the cursor should be positioned on a DRDA rowset. For rowset cursors, QRYROWNBR indicates that the cursor should be positioned on an SQL rowset.

QRYROWNBR is used in conjunction with the Query Scroll Orientation (QRYSCRORN) parameter specified on a CNTQRY command.

When used with a QRYSCRORN value of QRYSCRREL (relative), it identifies that the row (or start of the rowset) is calculated from the current row (or start of the current rowset) in the result table.

When used with a QRYSCRORN value of QRYSCRABS (absolute), it identifies that the row or start of the rowset is calculated from the beginning of the result table (if the QRYROWNBR is positive) or from the end of the result table (if the QRYROWNBR is negative).

When used with other QRYSCRORN values, the QRYROWNBR value is ignored.

If an application requester cannot convert a row number value provided by the application because it is larger than the largest number that can be transmitted in this instance variable, the application requester fails the request with an SQLSTATE of 56051.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	12	
class	X'213D'	
value	INSTANCE_OF LENGTH NOTE	BINDR - Binary Number Field 64 For QRYSCRORN of QRYSCRREL, and a QRYROWSET value of 1 or more, a value of -N...+N repositions the cursor to the Nth row before (-N) or after (+N) the current row and

then returns the rowset of size QRYROWSET starting at the new position.

If the QRYSCRREL succeeds, the following mappings are in effect:

SPCVAL	0: return the current row (non-rowset cursor) 0 returns the rowset of size QRYROWSET starting with the current row (rowset cursor).
SPCVAL	+1: return the next row (non-rowset cursor) +1 repositions the cursor forward one row, and returns the rowset of size QRYROWSET starting at the new position (rowset cursor).
SPCVAL	-1: return the prior row (non-rowset cursor) -1 repositions the cursor backward one row, and returns the rowset of size QRYROWSET starting at the new position (rowset cursor).
DFTVAL	+1: return the next row (non-rowset cursor) +1 repositions the cursor forward one row, and returns the rowset of size QRYROWSET starting at the new position (rowset cursor).
NOTE	For QRYSCRORN of QRYSCRABS, a value of +1...+N returns the row starting with the Nth row from the beginning of the result table. A value of -1...-N returns the row starting with the Nth row from the end of the result table (non-rowset cursor).  For QRYSCRORN of QRYSCRABS, a value of +1...+N returns the rowset of size QRYROWSET starting with the Nth row from the beginning of the result table. A value of -1...-N returns the rowset of size QRYROWSET starting with the Nth row from the end of the result table (rowset cursor).  If the QRYSCRABS succeeds, the following mappings are in effect:
SPCVAL	0: return the current row (non-rowset cursor) 0: returns the rowset of size QRYROWSET starting with the current row (rowset cursor)
SPCVAL	-1: return the last row (non-rowset cursor) -k (where k is the value of QRYROWSET): return the last rowset ending with the last row (rowset cursor)
DFTVAL	+1: return the first row (non-rowset cursor) +1 returns the rowset of size QRYROWSET starting with the first row (rowset cursor)
MINLVL	7

---

<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

---

**SEE ALSO**

<b>insvar</b>	<i>CNTQRY</i> (on page 222) <i>QRYSCORN</i> (on page 710)
<b>Semantic</b>	<i>QRYROWSET</i> (on page 697) <i>QRYROWNBR</i> (on page 694)

## NAME

QRYROWSET — Query Rowset Size

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'2156'  
**Length** \*  
**Class** CLASS  
**Sprcls** BIN - Binary Integer Number

Query Rowset Size (QRYROWSET) on CNTQRY command for cursors or result sets enabled for rowset positioning provides support for a multi-row fetch (an atomic operation). This parameter on an OPNQRY, CNTQRY, or EXCSQLSTT command indicates whether a DRDA rowset is to be returned with the command for a non-rowset cursor or result set. The effect on an OPNQRY, CNTQRY, and EXCSQLSTT for these queries 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.

For an OPNQRY command, a DRDA rowset of size *S* consists of the first row in the result table followed by the next *S*-1 rows in sequence (FETCH NEXT ROWSET) in the result table. (This is only for DRDA rowsets, and not for SQL rowsets.) For a CNTQRY command, a DRDA rowset of size *S* consists of the DRDA rowset fetched using the QRYSCRORN and QRYROWNBR in the CNTQRY followed by the next *S*-1 rows in sequence (FETCH NEXT ROWSET<sup>12</sup>) in the result table. QRYROWSET on an EXCSQLSTT command for a stored procedure call indicates whether a DRDA rowset of rows is to be returned for any non-dynamic scrollable cursors, non-rowset cursors conforming to the limited block query protocol returned by the stored procedure. For each such result set returned by the stored procedure, a DRDA rowset of size *S* consists of the first row in the result table for the result set followed by the next *S*-1 rows in sequence (FETCH NEXT ROWSET) in the result table.

For limited block protocol, the DRDA rowset is said to be complete when the requested number of rows (*S*) are fetched or when a fetch results in a negative SQLSTATE or an SQLSTATE of 02000. Otherwise, the DRDA rowset is incomplete and it is the source SQLAM's responsibility to dispose of the incomplete DRDA rowset as indicated below. Generally speaking, with the exception of a very large row size relative to the query block size or a very large QRYROWSET value, a single CNTQRY request will result in a complete DRDA rowset.

QRYROWSET applies to non-dynamic scrollable, non-rowset cursors and to non-scrollable, non-rowset cursors conforming to the limited block query protocol.

QRYROWSET applies to rowset scrollable cursors and to non-scrollable cursors conforming to the fixed row protocol for all rowset-enabled cursors.

For a non-scrollable cursor or result set, the following applies:

- If the QRYPRCTYP is FIXROWPRC, the QRYROWSET is ignored on all subsequent CNTQRY commands on non-rowset-enabled cursors.

---

12. FETCH NEXT is equivalent to FETCH NEXT ROWSET for a rowset value of 1.

- If the QRYPRCTYP is LMTBLKPRC:
  - If QRYROWSET is not specified on the OPNQRY command or EXCSQLSTT command for a stored procedure call, then the QRYROWSET parameter is not allowed on any CNTQRY command for the cursor or result set.
  - If QRYROWSET is specified on the OPNQRY command or EXCSQLSTT command for a stored procedure call, then the QRYROWSET parameter is required on every CNTQRY command for the cursor or result set.

If a QRYROWSET value is not specified for a non-dynamic scrollable cursor, then the following behavior applies for the cursor according to the QRYPRCTYP parameter returned on the OPNQRYRM:

- If the QRYPRCTYP is FIXROWPRC:
  - QRYROWSET is required on the CNTQRY command for all rowset cursors.
  - For non-rowset cursors, if QRYROWSET is not specified on the OPNQRY command, then no rows are returned with the command.
  - For non-rowset cursors, if QRYROWSET is not specified on a CNTQRY command, then no more than one FETCH request is performed for the CNTQRY command.
  - Dynamic scrollable cursors always have a QRYPRCTYP of FIXROWPRC. QRYROWSET is not allowed since it is not possible to manage cursor positions by row number for such cursors.
- If the QRYPRCTYP is LMTBLKPRC:
  - If QRYROWSET is not specified on the OPNQRY command, then an implicit DRDA rowset is returned. The implicit DRDA rowset has a size that is defined by DRDA Query Data Transfer Protocol rule QP4 and does not include any extra query blocks.
  - If QRYROWSET is not specified on the first CNTQRY command, then the cursor will be used in a non-scrolling manner and QRYROWSET may not be specified on subsequent CNTQRY commands. The behavior for all CNTQRY commands is as for Limited Block Protocol for non-scrollable cursors.
  - If QRYROWSET is specified on the first CNTQRY command, then a QRYROWSET value must be specified on every subsequent CNTQRY command, even if the value is 1. This parameter is a signal that the application requester will be using the cursor in a scrolling manner.

If a QRYROWSET value is specified on the CNTQRY command, the following applies:

- Specification of QRYROWSET means that the application requester must also specify a RTNEXTDTA value of RNTEXTALL if LOBs are to be returned in the answer set. This facilitates management of rowsets across several data server sites and also simplifies any query block reset operations (QRYBLKRST) when incomplete rowsets need to be terminated. The application server issues PRCCNVRM if this is not so.
- A QRYROWSET value of zero on the OPNQRY and EXCSQLSTT commands instructs the server to return no rows with the OPNQRYRM for the cursor or result set. A QRYROWSET value of zero (0) on the CNTQRY command is invalid for non-rowset cursors and results in the application server generating a SYNTAXRM with a SYNERCD value of '15'x (reserved value not allowed).
- For limited block protocol, a positive QRYROWSET value of *S* is valid on the OPNQRY, CNTQRY, and EXCSQLSTT commands. It instructs the application server to return no more



than *S* rows for the cursor or result set, taking extra query block limits into account. The application server stops fetching when the DRDA rowset is complete. The application server may also stop fetching if it encounters extra query block limits leaving the DRDA rowset in an incomplete state. The application requester is responsible for disposing of the incomplete DRDA rowset by issuing another CNTQRY that either:

- Requests the remaining rows in the incomplete DRDA rowset, by specifying a new QRYROWSET value equal to the number of rows remaining in the original DRDA rowset and a QRYSCRORN/QRYROWNBR resulting in a FETCH NEXT orientation.
- or:
- Terminates the incomplete DRDA rowset, by specifying a QRYBLKRST value of TRUE.

Only complete rows are used for this tracking—if a partial row is sent, the row is not counted until the remainder of the row is sent by the application server.

- A negative QRYROWSET value is invalid and results in the application server generating a SYNTAXRM with a SYNERRCD value of '15'x (reserved value not allowed).
- Any fetch that results only in setting the cursor to a desired position without returning data is known as a *positioning fetch*. This can occur if a CNTQRY request specifies a QRYRTNDTA of FALSE or if the QRYSCRORN parameter in conjunction with the QRYROWNBR parameter by definition results only in setting the cursor position without returning row data (for example, FETCH BEFORE, FETCH AFTER). For a positioning fetch, the QRYROWSET value is ignored and the DRDA rowset is considered complete after the positioning fetch is done.
- The QRYROWSNS parameter on a CNTQRY request applies to all rows fetched by the CNTQRY command.
- The OUTOVR object with a CNTQRY request applies to all rows fetched by the CNTQRY command.

If a QRYROWSET value is specified on the OPNQRY, CNTQRY, or EXCSQLSTT command for a non-rowset cursor, then the position of the cursor or result set as known by the target SQLAM may be different from the current position at the source SQLAM. It is the responsibility of the source SQLAM to manage these differences in cursor position. See the DRDA Reference, Appendix B (Scrollable Cursors) for more details on how to manage these cursor position differences.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	8	
class	X'2156'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	32
	MINVAL	0, for OPNQRY
	MINVAL	1, for CNTQRY
	MAXVAL	32,767
	MINLVL	7
<b>clscmd</b>	<b>NIL</b>	

---

**inscmd**                    **NIL**

**SEE ALSO**

**insvar**                    *BINDR* (on page 120)  
                               *CNTQRY* (on page 222)  
                               *EXCSQLSTT* (on page 381)  
                               *NBRROW* (on page 530)  
                               *OPNQRY* (on page 555)  
                               *QRYROWNBR* (on page 694)

**Semantic**                *BIN* (on page 118)  
                               *CLASS* (on page 158)  
                               *CNTQRY* (on page 222)  
                               *EXCSQLSTT* (on page 381)  
                               *FIXROWPRC* (on page 417)  
                               *LMTBLKPRC* (on page 475)  
                               *NBRROW* (on page 530)  
                               *OPNQRY* (on page 555)  
                               *OPNQRYRM* (on page 566)  
                               *PRCCNVRM* (on page 625)  
                               *QDDRDBD* (on page 657)  
                               *QRYBLKRST* (on page 677)  
                               *QRYPRCTYP* (on page 692)  
                               *QRYROWNBR* (on page 694)  
                               *QRYSCRCUR* (on page 706)  
                               *QRYSCRFST* (on page 707)  
                               *QRYSCRLST* (on page 708)  
                               *QRYSCRNXT* (on page 709)  
                               *QRYSCRORN* (on page 710)  
                               *QRYSCRPRI* (on page 712)  
                               *RTNEXTALL* (on page 791)  
                               *RTNEXTDTA* (on page 792)  
                               *SQLAM* (on page 847)  
                               *SYNERRCD* (on page 985)  
                               *SYNTAXRM* (on page 989)

**NAME**

QRYROWSNS — Query Row Sensitivity

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2153'  
**Length** \*  
**Class** CLASS  
**Sprcls** BOOLEAN - Logical Value

Query Row Sensitivity (QRYROWSNS) on a CNTQRY command indicates whether the rows to be fetched are to be sensitive to changes in the data underlying the rows in the result table. The relational database at the target server determines whether the requested sensitivity is compatible with the defined sensitivity of the cursor and fails the request if it is incompatible.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	5	
class	X'2153'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	8
	ENUVAL	X'F1' - TRUE - True State
	NOTE	The rows fetched are to be sensitive to changes made to rows underlying the result table after it has been materialized
	ENUVAL	X'F0' - FALSE - False State
	NOTE	The rows fetched are to be insensitive to changes made to rows underlying the result table after it has been materialized
	DFTVAL	X'F1' - TRUE if the QRYATTSNS value returned on the OPNQRYRM for the cursor indicates the cursor is sensitive. Otherwise, X'F0'.
	MINLVL	7
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** CNTQRY (on page 222)

**NAME**

QRYRTNDDTA — Query Returns Data

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2155'  
**Length** \*  
**Class** CLASS  
**Sprcls** BOOLEAN - Logical Value

Query Returns Data (QRYRTNDDTA) on a CNTQRY command indicates whether data is to be returned with the command.

The value of FALSE on the CNTQRY command results only in a setting of the cursor position based on the QRYSCRORN and QRYROWNBR parameters on the CNTQRY command.

See also *QRYROWSET* (on page 697) for interactions with this parameter.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	5	
class	X'2155'	
value	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	LENGTH	2
	ENUVAL	X'F1' - TRUE - True State
	NOTE	Data is to be returned with the command.
	ENUVAL	X'F0' - FALSE - False State
	NOTE	Data is not to be returned with the command.
	DFTVAL	X'F1' - TRUE - True State
	MINLVL	7
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** CNTQRY (on page 222)

**NAME**

QRYSCRABS — Query Scroll Absolute Orientation

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Length** \*

**Class** CONSTANT

On a CNTQRY, the QRYSCRABS value in the QRYSCRORN parameter indicates that the orientation for the fetch operation(s) is absolute.

The CNTQRY causes the row specified by its absolute row number to be fetched. If a QRYROWSET value is specified, the next rows in sequence after that row are fetched, according to the description in the DDM term QRYROWSET.

This parameter may be used in conjunction with the QRYROWNBR parameter.

---

value	2
-------	---

**SEE ALSO**

- insvar**
  - CNTQRY (on page 222)
  - QRYROWNBR (on page 694)
  - QRYSCRORN (on page 710)
- Semantic**
  - QRYROWNBR (on page 694)

**NAME**

QRYSCRAFT — Query Scroll After Orientation

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Length** \*

**Class** CONSTANT

On a CNTQRY, the QRYSCRAFT value in the QRYSCRORN parameter indicates that the orientation for the fetch operation(s) is **after**, and that the cursor position is to be placed after the last row in the result table.

The SQLSTATE returned for this operation is 00000.

---

value	3
-------	---

**SEE ALSO**

**insvar**            *CNTQRY* (on page 222)  
                      *QRYSCRORN* (on page 710)

**NAME**

QRYSCRBEF — Query Scroll Before Orientation

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Length** \***Class** CONSTANT

On a CNTQRY, the QRYSCRBEF value in the QRYSCRORN parameter indicates that the orientation for the fetch operation(s) is **before**, and that the cursor position is to be placed before the first row in the result table.

The SQLSTATE returned for this operation is 00000.

---

 value

4

**SEE ALSO****insvar** QRYSCRORN (on page 710)

**NAME**

QRYSCRCUR — Query Scroll Current Orientation

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Length** \*  
**Class** CONSTANT

On a CNTQRY, the QRYSCRCUR value in the QRYSCRORN parameter indicates that the orientation for the fetch operation(s) is **current**. The CNTQRY causes the current row to be fetched if a QRYROWSET value is not specified. If a QRYROWSET value is specified, the current rows in sequence in the result table are fetched, according to the description in QRYROWSET. The type of rowset returned when a QRYROWSET value is specified depends on whether the cursor is a non-rowset cursor (return DRDA rowset) or a rowset cursor (return SQL rowset). See the DRDA Reference for more details.

---

value	8
-------	---

**SEE ALSO**

**insvar** QRYSCRORN (on page 710)



**NAME**

QRYSCRFST — Query Scroll First Orientation

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Length** \***Class** CONSTANT

On a CNTQRY, the QRYSCRFST value in the QRYSCRORN parameter indicates that the orientation for the fetch operation(s) is **first**. The CNTQRY causes the first row to be fetched if a QRYROWSET value is not specified. If a QRYROWSET value is specified, the first rows in sequence in the result table are fetched, according to the description in QRYROWSET. The type of rowset returned when a QRYROWSET value is specified depends on whether the cursor is a non-rowset cursor (return DRDA rowset) or a rowset cursor (return SQL rowset). See the DRDA Reference for more details.

---

 value

6

**SEE ALSO****Semantic** QRYSCRORN (on page 710)

**NAME**

QRYSCRLST — Query Scroll Last Orientation

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Length** \*

**Class** CONSTANT

On a CNTQRY, the QRYSCRLST value in the QRYSCRORN parameter indicates that the orientation for the fetch operation(s) is **last**. The CNTQRY causes the last row to be fetched if a QRYROWSET value is not specified. If a QRYROWSET value is specified, the last rows in sequence in the result table are fetched, according to the description in QRYROWSET. The type of rowset returned when a QRYROWSET value is specified depends on whether the cursor is a non-rowset cursor (return DRDA rowset) or a rowset cursor (return SQL rowset). See the DRDA Reference for more details.

---

value

7

**SEE ALSO**

**insvar** QRYSCRORN (on page 710)

**NAME**

QRYSCRNXT — Query Scroll Next Orientation

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Length** \***Class** CONSTANT

On a CNTQRY, the QRYSCRNXT value in the QRYSCRORN parameter indicates that the orientation for the fetch operation(s) is **next**. The CNTQRY causes the next row to be fetched if a QRYROWSET value is not specified. If a QRYROWSET value is specified, the next rows in sequence in the result table are fetched, according to the description in QRYROWSET. The type of rowset returned when a QRYROWSET value is specified depends on whether the cursor is a non-rowset cursor (return DRDA rowset) or a rowset cursor (return SQL rowset). See the DRDA Reference for more details.

---

value	0
-------	---

**SEE ALSO****insvar** QRYSCRORN (on page 710)

**NAME**

QRYSCRORN — Query Scroll Orientation

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2152'  
**Length** \*  
**Class** CLASS  
**Sprcls** BIN - Binary Integer Number

Query Scroll Orientation (QRYSCRORN) controls the scroll orientation of a scrollable cursor.

The meaning of QRYSCRORN depends on whether the cursor is a rowset cursor or a non-rowset cursor, and upon the value of QRYROWSET. If QRYROWSET is specified, then a rowset operation is being performed. If QRYROWSET is not specified, then a single row operation is being performed. See the DRDA Reference, Section 1.1.2 for a discussion of the different cursor types. For non-rowset cursors without a value specified for QRYROWSET, QRYSCRORN indicates that the cursor should be positioned on a single row according to the value specified (see enumerated values below). For non-rowset cursors with a value specified for QRYROWSET, QRYSCRORN indicates that the cursor should be positioned on a DRDA rowset according to the value specified. For rowset cursors, QRYSCRORN indicates that the cursor should be positioned on an SQL rowset according to the value specified.

QRYSCRORN may be used in conjunction with the QRYROWNBR parameter specified on the CNTQRY command.

clsvar	NIL	
insvar	CLASS INSTANCE VARIABLES	
length	5	
class	X'2152'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	8
	ENUVAL	0 QRYSCRNXT
	NOTE	The fetch orientation is <b>next</b> , where current cursor position is to be set to the next row or rows of the result table relative to the current cursor position.
	ENUVAL	1 QRYSCRREL
	NOTE	The fetch orientation is <b>relative</b> , where QRYROWNBR is the value indicating the position of the desired row relative to the current cursor position.
	ENUVAL	2 QRYSCRABS
	NOTE	The fetch orientation is <b>absolute</b> , where QRYROWNBR is the value indicating the absolute position of the desired row.
	ENUVAL	3 QRYSCRAFT

	NOTE	The fetch orientation is <b>after</b> , where current cursor position is to be set <b>after</b> the last row in the result table for the cursor.
	ENUVAL NOTE	4 QRYSCRBEF The fetch orientation is <b>before</b> , where current cursor position is to be set before the first row in the result table for the cursor.
	ENUVAL NOTE	5 QRYSCRPRI The fetch orientation is <b>prior</b> , where current cursor position is to be set to the prior (that is, previous) row or rows of the result table relative to the current cursor position.
	ENUVAL NOTE	6 QRYSCRFST The fetch orientation is <b>first</b> , where current cursor position is to be set to the first row of the result table.
	ENUVAL NOTE	7 QRYSCRLST The fetch orientation is <b>last</b> , where current cursor position is to be set to the last row of the result table.
	ENUVAL NOTE	8 QRYSCRCUR The fetch orientation is <b>current</b> , where cursor position is to be set to the current row of the result table (that is, cursor position is unchanged) or the first row of the current rowset, depending on the number of rows specified.
	DFTVAL MINLVL	0 QRYSCRNXT 7
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

<b>insvar</b>	<i>CNTQRY</i> (on page 222) <i>QRYROWNBR</i> (on page 694)
<b>Semantic</b>	<i>QRYROWNBR</i> (on page 694) <i>QRYROWSET</i> (on page 697) <i>QRYSCRCUR</i> (on page 706) <i>QRYSCRFST</i> (on page 707) <i>QRYSCRLST</i> (on page 708) <i>QRYSCRNXT</i> (on page 709) <i>QRYSCRPRI</i> (on page 712)

**NAME**

QRYSCRPRI — Query Scroll Prior Orientation

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Length** \*

**Class** CONSTANT

On a CNTQRY, the QRYSCRPRI value in the QRYSCRORN parameter indicates that the orientation for the fetch operation(s) is **prior**. The CNTQRY causes the prior (that is, previous) row to be fetched if a QRYROWSET value is not specified. If a QRYROWSET value is specified, the prior rows in sequence in the result table are fetched (up to and including the previous row), according to the description in QRYROWSET. The type of rowset returned when a QRYROWSET value is specified depends on whether the cursor is a non-rowset cursor (return DRDA rowset) or a rowset cursor (return SQL rowset). See the DRDA Reference for more details.

---

value	5
-------	---

**SEE ALSO**

**insvar**            *QRYSCRORN* (on page 710)

**NAME**

QRYSCRREL — Query Scroll Relative Orientation

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Length** \*  
**Class** CONSTANT

On a CNTQRY, the QRYSCRREL value in the QRYSCRORN parameter indicates that the orientation for the fetch operation(s) is relative. The CNTQRY fetches the row whose row number is relative to the current cursor position by the value in QRYROWNBR. For example, if QRYROWNBR is -2, then the row that is two rows previous to the current cursor position is fetched. If a QRYROWSET value is specified, the next rows in sequence after that row are fetched, according to the description in the DDM term QRYROWSET.

This parameter may be used in conjunction with the QRYROWNBR parameter.

---

value	1
-------	---

**SEE ALSO**

**insvar** CNTQRY (on page 222)  
 QRYROWNBR (on page 694)  
 QRYSCRORN (on page 710)  
**Semantic** QRYROWNBR (on page 694)

**NAME**

QRYSNSDYN — Query Sensitive with Dynamic Membership

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Length** \*

**Class** CONSTANT

On an OPNQRYRM, the QRYSNSDYN value in the QRYATTSNS parameter indicates that the query being opened is sensitive to changes to the underlying rows of the result table. This includes changes made through the cursor by the application as well as committed changes made by other processes. In addition, the scrollable sensitive cursor being opened has a result table which dynamically changes size and ordering with updates to the base table.

See the DRDA Reference, Appendix B for an overview of scrollable cursors.

---

value	3
-------	---

**SEE ALSO**

- Semantic**
- EXCSQLSTT* (on page 381)
  - LMTBLKPRC* (on page 475)
  - OPNQRY* (on page 555)
  - OPNQRYRM* (on page 566)
  - QRYATTSNS* (on page 667)



**NAME**

QRYSNSSTC — Query Sensitive with Static Membership

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Length** \***Class** CONSTANT

On an OPNQRYRM, the QRYSNSDYN value in the QRYATTSNS parameter indicates that the query being opened is sensitive to changes to the underlying rows of the result table. This includes changes made through the cursor by the application as well as committed changes made by other processes. In addition, the scrollable sensitive cursor being opened has a result table with a static size and static ordering of rows.

See the DRDA Reference, Appendix B for an overview of scrollable cursors.

---

value	2
-------	---

**SEE ALSO**

**Semantic**      *EXCSQLSTT* (on page 381)  
                   *OPNQRY* (on page 555)  
                   *OPNQRYRM* (on page 566)  
                   *QRYATTSNS* (on page 667)

**NAME**

QRYUNK — Query Attribute is Unknown or Undefined

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Length** \*

**Class** CONSTANT

On an OPNQRYRM, the QRYUNK value in the QRYATTxxx parameter for which it is an enumerated value indicates that the query being opened does not have a defined value for this attribute or the value is unknown.

---

value	0
-------	---

**SEE ALSO**

- Semantic**
- EXCSQLSTT* (on page 381)
  - OPNQRY* (on page 555)
  - OPNQRYRM* (on page 566)
  - QRYATTSNS* (on page 667)
  - QRYATTUPD* (on page 669)

**NAME**

QRYUPD — Query Deletable and Updatable

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Length** \***Class** CONSTANT

On an OPNQRYRM, the QRYUPD value in the QRYATTUPD parameter indicates that the query being opened allows read, delete, and update operations on the cursor. Each such delete or update operation can cause the base table to be modified and applies to a row that has been fetched.

---

 value

4

**SEE ALSO**

**Semantic**      *EXCSQLSTT* (on page 381)  
                   *OPNQRY* (on page 555)  
                   *OPNQRYRM* (on page 566)  
                   *QRYATTUPD* (on page 669)

## NAME

RDB — Relational Database

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD**Codepoint** X'240F'**Length** \***Class** CLASS**Sprcls** MANAGER - Resource Manager

Relational Database (RDB) is a database in which data is stored as a collection of tables. The programming that implements the class commands and instance commands of class Relational Database can be viewed as a *database management system*. Instances of class RDB might or might not exist in a DDM server. All of them are instances of class RDB. The DDM architecture does not provide models for either the structure of RDB instances or for the commands class RDB implements.

SQL is the language application programs use to access and to modify data in a relational database. See *SQL* (on page 835) for a discussion of the relationship of SQL and the DDM architecture.

Among other things, some of the objects a relational database can contain are:

- A catalog containing information about data, storage, collections, packages, and authorization
- Tables consisting of columns and rows

At the intersection of every column and row is a specific item of data that is an instance of an *SQL data type*.

- Indexes that are ordered sets of pointers to the data in relational tables

Each index is based on the values of data in one or more table columns. Indexes improve access performance and ensure uniqueness of row keys.

- Views that provide an alternate way of looking at the data in one or more table

Views help to improve access performance.

- Locks that prevent one program from accessing data that another program has changed but has not yet committed to the RDB

Lock management is an internal concern of the database that the DDM architecture does not address.

- Recovery logs containing the information necessary for performing unit of work commit and rollback functions
- Packages that define how an application program interacts with the relational database

Items associated with packages are:

1. Definitions of the data variables in which the application receives data items from the RDB or from which the application sends data items to the RDB
2. The SQL statements that are executed when the application requests

3. Numbered *sections* containing algorithms that direct the database processing one or more SQL statements require

Either none or the required number of sections are reserved for the binding of dynamic SQL statements.

4. A *consistency token* which the RDB checks on all subsequent accesses through the package

This token ensures that the application and the package remain consistent over time because changes might occur to the application or to the RDB.

- Authorizations which define a user's rights to access or alter data or to administer the database
- Cursors which allow an application program to point to a row of interest
- RDB collections which are named user-defined collections of packages

See *RDBOVR* (on page 740) for a discussion on how DDM facilities access a remote RDB.

The RDB can share recoverable resources and locks across a set of SYNCPTMGR protected connections that are identified by the same XID. The XID and the XIDSHR parameters are passed to the RDB by the SYNCPTMGR. If the XIDSHR indicator is partial, then the RDB compares the XID of all SYNCPTMGR protected connections and shares resources and locks with the set of connections whose XID matches exactly. If complete sharing is required, the RDB only compares the GTRID part of the XID, and shares resources and locks between the set of connection whose GTRID part of the XID matches. The XID parameter and XAFLAG(TM\_LCS) flag will be transmitted to the RDB by the XAMGR for resource sharing. The flag TM\_LCS 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, then it should ignore the flag TM\_LCS and revert to its default behavior regarding locks (see *XAMGROV* (on page 1066) for more details).

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'240F'	
typdef	INSTANCE_OF	TYPDEF - Data Type to Data Representation Definition
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	
<b>mgrlvl</b>	3	
<b>mgrdepls</b>	NIL	
<b>vldattls</b>	VALID ATTRIBUTES	
X'0019'	INSTANCE_OF	HELP - Help Text
X'2110'	INSTANCE_OF	RDBNAM - Relational Database Name
X'0045'	INSTANCE_OF	TITLE - Title

#### SEE ALSO

<b>clscmd</b>	<i>SQLAM</i> (on page 847)
<b>cmddta</b>	<i>EXCSQLSET</i> (on page 377)

<b>cmdrpy</b>	<i>ACCSEC</i> (on page 52) <i>CNTQRY</i> (on page 222)
<b>inscmd</b>	<i>SQLAM</i> (on page 847)
<b>insvar</b>	<i>ACCSEC</i> (on page 52) <i>BNDSQLSTT</i> (on page 136) <i>CLSQRY</i> (on page 165) <i>CMDCHKRM</i> (on page 173) <i>CNTQRY</i> (on page 222) <i>DECPRC</i> (on page 266) <i>DFTRDBCOL</i> (on page 274) <i>DGRIOPRL</i> (on page 279) <i>DRPPKG</i> (on page 293) <i>ENDBND</i> (on page 336) <i>EXCSQLSTT</i> (on page 381) <i>MGRLVL</i> (on page 506) <i>OBJNSPRM</i> (on page 542) <i>PKGNAME</i> (on page 594) <i>PKGNAMECSN</i> (on page 596) <i>PKGNAMECT</i> (on page 598) <i>PKGSNLST</i> (on page 607) <i>PRCCNVRM</i> (on page 625) <i>PRPSQLSTT</i> (on page 636) <i>QRYNOPRM</i> (on page 690) <i>QRYPOPRM</i> (on page 691) <i>RDBALWUPD</i> (on page 726) <i>RDBCMTOK</i> (on page 731) <i>REBIND</i> (on page 753) <i>RSCLMTRM</i> (on page 778) <i>RSLSETRM</i> (on page 785) <i>SECCHK</i> (on page 800) <i>STTSTRDEL</i> (on page 899) <i>SYNTAXRM</i> (on page 989) <i>VALNSPRM</i> (on page 1057)
<b>mgrdepls</b>	<i>SQLAM</i> (on page 847)
<b>Semantic</b>	<i>ABNUOWRM</i> (on page 39) <i>ACCDMG</i> (on page 41) <i>ACCRDB</i> (on page 42) <i>ACCRDBRM</i> (on page 48) <i>APPCMNFL</i> (on page 68) <i>BGNBND</i> (on page 110) <i>BNDCHKEXS</i> (on page 125) <i>BNDERRALW</i> (on page 128) <i>BNDEXSOPT</i> (on page 130) <i>BNDEXSRQR</i> (on page 131) <i>BNDNERALW</i> (on page 132) <i>BNDSQLSTT</i> (on page 136) <i>BNDSTTASM</i> (on page 141) <i>CLSQRY</i> (on page 165) <i>CMNAPPC</i> (on page 184) <i>CMNSYNCPT</i> (on page 202)

*CMNTCPIP* (on page 214)  
*CNTQRY* (on page 222)  
*CSTBITS* (on page 247)  
*CSTMBCS* (on page 248)  
*CSTSBCS* (on page 249)  
*CSTSYSDFT* (on page 250)  
*DFTRDBCOL* (on page 274)  
*DRPPKG* (on page 293)  
*DSCRDBTBL* (on page 300)  
*DSCSQLSTT* (on page 304)  
*DTAOVR* (on page 321)  
*ENDBND* (on page 336)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSET* (on page 377)  
*EXCSQLSTT* (on page 381)  
*FIXROWPRC* (on page 417)  
*INHERITANCE* (on page 437)  
*MANAGER* (on page 491)  
*OPNQRY* (on page 555)  
*PKGBNARM* (on page 585)  
*PKGBPARM* (on page 586)  
*PKGCNSTKN* (on page 588)  
*PKGID* (on page 591)  
*PKGISOLVL* (on page 592)  
*PKGAMCSN* (on page 596)  
*PKGOWNID* (on page 600)  
*PKGSNLST* (on page 607)  
*PRPSQLSTT* (on page 636)  
*RDBAFLRM* (on page 725)  
*RDBALWUPD* (on page 726)  
*RDBATHRM* (on page 727)  
*RDBCMM* (on page 728)  
*RBCMTOK* (on page 731)  
*RBCOLID* (on page 732)  
*RDBNACRM* (on page 734)  
*RDBNAM* (on page 736)  
*RDBNFNRM* (on page 739)  
*RDBOVR* (on page 740)  
*RDBRLLBCK* (on page 745)  
*RDBRLSCMM* (on page 748)  
*RDBRLSCNV* (on page 749)  
*RDBRLSOPT* (on page 750)  
*REBIND* (on page 753)  
*RSCLMTRM* (on page 778)  
*RSYNCMGR* (on page 787)  
*SQL* (on page 835)  
*SQLAM* (on page 847)  
*SQLCARD* (on page 855)  
*SQLDTARD* (on page 862)  
*SQLOBJNAM* (on page 866)  
*SQLSTT* (on page 868)  
*SRVLST* (on page 878)

*SUBSETS* (on page 902)  
*SYNCCTL* (on page 915)  
*SYNCPTMGR* (on page 939)  
*SYNCPTOV* (on page 944)  
*TCPCMNFL* (on page 992)



**NAME**

RDBACCCL — RDB Access Manager Class

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'210F'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

RDB Access Manager Class (RDBACCCL) String specifies that the SQL application manager accesses the relational database.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	6	
class	X'210F'	
value	INSTANCE_OF REQUIRED	CODPNTDR - Codepoint Data Representation
	ENUVAL	X'2407' - SQLAM - SQL Application Manager
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** ACCRDB (on page 42)

**NAME**

RDBACCRM — RDB Currently Accessed

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2207'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

RDB Currently Accessed (RDBACCRM) Reply Message indicates that the ACCRDB command cannot be issued because the requester currently has access to a relational database.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'2207'	
<b>rdbnam</b>	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

- cmdrpy** ACCRDB (on page 42)  
INTRDBRQS (on page 445)
- Semantic** ACCRDB (on page 42)  
INTRDBRQS (on page 445)

**NAME**

RDBAFLRM — RDB Access Failed Reply Message

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'221A'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

RDB Access Failed (RDBAFLRM) Reply Message specifies that the relational database (RDB) failed the attempted connection.

An SQLCARD object must also be returned, following the RDBAFLRM, to explain why the RDB failed the connection. In addition, the target SQLAM instance is destroyed.

**DSS Carrier:** RPYDSS

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'221A'	
<b>rdbnam</b>	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code  8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**cmdrpy** ACCRDB (on page 42)  
ACCSEC (on page 52)

**rpydta** ACCRDB (on page 42)

**Semantic** ACCRDB (on page 42)

**NAME**

RDBALWUPD — RDB Allow Updates

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'211A'  
**Length** \*  
**Class** CLASS  
**Sprcls** BOOLEAN - Logical Value

RDB Allow Updates (RDBALWUPD) Boolean State controls whether the requester is allowed to perform any update operations in the relational database (RDB). An update operation is defined as a change to an RDB object in the target RDB such that the change to the RDB object is under commit or rollback control of the requester's current unit of work.

If the requester is not allowed to perform update operations in the RDB, then the requester is limited to read-only access to the RDB resources.

In addition, the target RDB must not allow the EXCSQLIMM or EXCSQLSTT command to execute an SQL commit or rollback.

The value TRUE indicates that the requester is allowed to update the RDB.

The value FALSE indicates that the requester is not allowed to update the RDB.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	5	
class	X'211A'	
value	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	LENGTH	2
	ENUVAL	X'F1' - TRUE - True State
	NOTE	Updates are allowed in the RDB.
	ENUVAL	X'F0' - FALSE - False State
	NOTE	Updates are not allowed in the RDB.
	DFTVAL	X'F1' - TRUE - True State
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** ACCRDB (on page 42)

**NAME**

RDBATHRM — Not Authorized to RDB

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2203'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

Not Authorized to RDB (RDBATHRM) Reply Message specifies that the requester is not authorized to access the specified relational database.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'2203'	
<b>rdbnam</b>	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**cmdrpy** ACCRDB (on page 42)

**Semantic** ACCRDB (on page 42)

**NAME**

RDBCMM — RDB Commit Unit of Work

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'200E'**Length** \***Class** CLASS**Sprcls** COMMAND - Command

RDB Commit Unit of Work (RDBCMM) Command commits all work performed for the current unit of work. The current unit of work ends and a new unit of work begins.

**Source System Processing**

The source system determines the location of the RDB:

Local            Call the local RDB server.

Remote          Send the RDBCMM command to the remote RDB server.

If the connection is to be reused at the completion of the commit or rollback, the release connection parameter (*rlsconv*) is set to REUSE. The connection cannot be reused until the receipt of the ENDUOWRM. If the target server responds with *rlsconv* set to FALSE, the connection is not allowed to be reused until the application terminates. If set to REUSE, the target server returns a set of SQLSTT objects to establish the application execution environment.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**DSS Carrier: RQSDSS****Target System Processing**

The *rdbnam* parameter specifies the name of the RDB which the ACCRDB command accessed. If the *rdbnam* parameter is specified, its value must be the same as that specified for the *rdbnam* parameter on the ACCRDB command.

If an RDBCMM occurs after a BGNBND and before the package binding process is terminated, then the binding process is implicitly terminated before the unit of work is committed.

If an RDBCMM occurs after an OPNQRY and before the query is terminated, then the query process is implicitly closed before the unit of work is committed.

The TYPDEFNAM reply data object specifies the name of the data type to data representation mapping definition that the target SQLAM uses when sending the SQLCARD object for this command.

The TYPDEFOVR reply data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type to data representation mapping definitions for the SQLCARD object for this command.

Normal completion of this command results in an ENDUOWRM being sent prior to any reply data object.

**Exceptions**

Exception conditions the RDB detects are reported in an SQLCARD object.

If the specified RDB is not currently accessed, then the command is rejected with the RDBNACRM.

If the CMNSYNCPT is being used, then the RDBCMM command is rejected with the CMDVLTRM.

**Source System Reply Processing**

Return any reply messages and data objects to the requester.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'200E'	
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
<b>clscmd</b>	NIL	
<b>cmddta</b>	NIL	
<b>inscmd</b>	NIL	
rlsconv	INSTANCE_OF ENUVAL ENUVAL NOTE  DFTVAL OPTIONAL MINLVL	RLSCONV - Release Connection X'F0'(NO) - Do not reuse connection X'F2' (REUSE) - Connection can be reused X'F1' (TERMINATE) is not allowed. If specified, it is a conversation protocol error.  X'F0' (NO) - Do not reuse connection 7
<b>rpydta</b>	SQLSTT	<b>REPLY OBJECTS</b>
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL MINLVL ENUVAL MINLVL DFTVAL NOTE  OPTIONAL	TYPDEFNAM - Data Type Definition Name 'QTDSQL370' 'QTDSQL400' 'QTDSQLX86' 'QTDSQLASC' 4 'QTDSQLVAX' 4 " The default means the value received on ACCRDBRM is used.

X'0035'	INSTANCE_OF OPTIONAL DFTVAL NOTE	TYPDEFOVR - TYPDEF Overrides " The default means the value received on ACCRDBRM is used.
X'1C03'	INSTANCE_OF OPTIONAL MINLVL	MGRLVLOVR - Manager Level Overrides 7
X'2408'	INSTANCE_OF REQUIRED	SQLCARD - SQL Communications Area Reply Data
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'220D'	INSTANCE_OF	ABNUOWRM - Abnormal End Unit of Work Condition
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported
X'221D'	INSTANCE_OF MINLVL	CMDVLTRM - Command Violation 4
X'220C'	INSTANCE_OF	ENDUOWRM - End Unit of Work Condition
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'2204'	INSTANCE_OF	RDBNACRM - RDB Not Accessed
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limits Reached
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'125F'	INSTANCE_OF	TRGNSPRM - Target Not Supported
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

**SEE ALSO**

<b>inscmd</b>	<i>SQLAM</i> (on page 847)
<b>Semantic</b>	<i>CMDVLTRM</i> (on page 181) <i>INTRDBRQS</i> (on page 445) <i>OPNQRY</i> (on page 555) <i>SQL</i> (on page 835) <i>SQLAM</i> (on page 847) <i>SUBSETS</i> (on page 902) <i>XAMGROV</i> (on page 1066)



**NAME**

RDBCMTOK — RDB Commit Allowed

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD  
**Codepoint** X'2105'  
**Length** \*  
**Class** CLASS  
**Sprcls** BOOLEAN - Logical Value

RDB Commit Allowed (RDMCMTOK) specifies whether an RDB should allow the processing of any commit or rollback operations that occur during execution of a command.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	5	
class	X'2105'	
value	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	LENGTH	2
	ENUVAL	X'F1' - TRUE - True State
	NOTE	RDB should allow the processing of commits and rollbacks.
	ENUVAL	X'F0' - FALSE - False State
	NOTE	RDB should not allow the processing of commits and rollbacks.
	DFTVAL	X'F0' - FALSE - False State
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *EXCSQLIMM* (on page 371)  
*EXCSQLSTT* (on page 381)

## NAME

RDBCOLID — RDB Collection Identifier

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD**Codepoint** X'2108'**Length** \***Class** CLASS**Sprcls** FIELD - A Discrete Unit of Data

The RDB Collection Identifier (RDBCOLID) identifies a unique collection of objects contained in a relational database, and it is used for user-defined grouping.

Prior to DDM Level 7, the RDBCOLID has a fixed length of 18 characters, including right blank padding if the collection ID is less than 18 characters long. As of DDM Level 7, the RDBCOLID can accommodate a collection ID of up to 255 characters in length, and its format will vary depending on the length of the collection ID:

- Length of collection ID  $\leq$  18 characters

There is no change to the format of the RDBCOLID. The length of the RDBCOLID remains fixed at 18 which includes any right blank padding if necessary.

- Length of collection ID  $>$  18 characters

The length of the RDBCOLID is identical to the length of the collection ID. No right blank padding is required.

The CURRENT PACKAGE PATH value can override the RDBCOLID specified inside the PKGNAMCSN parameter on those commands which flow this parameter. See the DRDA Reference, Section 4.4.3, Overriding the Collection ID of a Package Name and Section 6.4.5, Package Collection Resolution for more details.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
name	INSTANCE_OF	CHRSTRDR - Character String Data Representation
	MINLEN	18
	MAXLEN	255
	MINLVL	7
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	
<b>vldattls</b>	NIL	

## SEE ALSO

**insvar** *CHRSTRDR* (on page 156)  
*PKGNAM* (on page 594)  
*PKGNAMCSN* (on page 596)  
*PKGNAMCT* (on page 598)

**Semantic** *SQL* (on page 835)

**NAME**

RDBINTTKN — RDB Interrupt Token

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2103'  
**Length** \*  
**Class** CLASS  
**Sprcls** STRING - String <QDDPRMD>

RDB Interrupt Token (RDBINTTKN) String specifies a target SQLAM; a generated token that the target SQLAM uses to interrupt the execution of a DDM command for a specific requester. The contents of the token are unarchitected and can differ for each target SQLAM.

To interrupt a DDM command, the requester must specify the interrupt token value that a previous ACCRDBRM returned.

<b>clsvar</b>	NIL	
<b>insvar</b>	INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'2103'	
<b>value</b>	INSTANCE_OF	BYTSTRDR - Byte String <QDDPRMD>
	MAXLEN	255
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *BYTSTRDR* (on page 146)  
*INTTKNRM* (on page 448)  
*QDDPRMD* (on page 654)  
**Semantic** *ACCRDBRM* (on page 48)  
*INTRDBRQS* (on page 445)  
*INTTKNRM* (on page 448)  
*QDDRDBD* (on page 657)  
*SQLAM* (on page 847)

NAME

RDBNACRM — RDB Not Accessed

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'2204'  
**Length** \*  
**Class** CLASS  
**Sprcls** RPYMSG - Reply Message

RDB Not Accessed (RDBNACRM) Reply Message indicates that the access relational database command (ACCRDB) was not issued prior to a command requesting RDB services.

DSS Carrier: RPYDSS

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'2204'	
rdbnam	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

SEE ALSO

**cmdrpy** *BGNBND* (on page 110)  
*BNDSQLSTT* (on page 136)  
*CLSQR*Y (on page 165)  
*CNTQRY* (on page 222)  
*DRPPKG* (on page 293)  
*DSCRDBTBL* (on page 300)  
*DSCSQLSTT* (on page 304)  
*ENDBND* (on page 336)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSET* (on page 377)  
*EXCSQLSTT* (on page 381)  
*OPNQRY* (on page 555)  
*PRPSQLSTT* (on page 636)  
*RDBCMM* (on page 728)  
*RDBRLLBCK* (on page 745)  
*REBIND* (on page 753)

**Semantic** *BGNBND* (on page 110)  
*BNDSQLSTT* (on page 136)

*CLSQR*Y (on page 165)  
*CNTQR*Y (on page 222)  
*DRPPK*G (on page 293)  
*DSCRDBTBL* (on page 300)  
*DSCSQLSTT* (on page 304)  
*ENDBND* (on page 336)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSET* (on page 377)  
*EXCSQLSTT* (on page 381)  
*FIXROWPRC* (on page 417)  
*LMTBLKPRC* (on page 475)  
*OPNQR*Y (on page 555)  
*PRPSQLSTT* (on page 636)  
*RDBCMM* (on page 728)  
*RDBRLLBCK* (on page 745)  
*REBIND* (on page 753)

## NAME

RDBNAM — Relational Database Name

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'2110'  
**Length** \*  
**Class** CLASS  
**Sprcls** MGRNAM - Manager Name

Relational Database Name (RDBNAM) specifies the name of a relational database (RDB) of the server. A server can have more than one RDB. The RDBNAM syntax is not validated.

Prior to DDM Level 7, the RDBNAM has a fixed length of 18 characters, including right blank padding if the RDB name is less than 18 characters long. As of DDM Level 7, the RDBNAM can accommodate an RDB name of up to 255 characters in length, and its format will vary depending on the length of the RDB name:

- Length of RDB name <= 18 characters

There is no change to the format of the RDB name. The length of the RDBNAM remains fixed at 18 which includes any right blank padding if necessary.

- Length of RDB name > 18 characters

The length of the RDB name is identical to the length of the RDB name. No right blank padding is required.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'2110'	
<b>value</b>	INSTANCE_OF	CHRSTRDR - Character String Data Representation
	MINLEN	18
	MAXLEN	255
	MINLVL	7
	REQUIRED	
	NOTE	The syntax of this field is not validated.
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

## SEE ALSO

**insvar** ABNUOWRM (on page 39)  
 ACCRDB (on page 42)  
 ACCSEC (on page 52)  
 AGNPRMRM (on page 65)  
 BGNBND (on page 110)  
 BGNBNDRM (on page 117)  
 BNDSQLSTT (on page 136)  
 CHRSTRDR (on page 156)  
 CLSQRY (on page 165)  
 CMDATHRM (on page 171)

	<i>CMDCHKRM</i> (on page 173)
	<i>CMDNSPRM</i> (on page 176)
	<i>CMDVLTRM</i> (on page 181)
	<i>CMMRQSRM</i> (on page 182)
	<i>CNTQRY</i> (on page 222)
	<i>DRPPKG</i> (on page 293)
	<i>DSCINVRM</i> (on page 298)
	<i>DSCRDBTBL</i> (on page 300)
	<i>DSCSQLSTT</i> (on page 304)
	<i>DTAMCHRM</i> (on page 320)
	<i>ENDBND</i> (on page 336)
	<i>ENDQRYRM</i> (on page 342)
	<i>ENDUOWRM</i> (on page 343)
	<i>EXCSQLIMM</i> (on page 371)
	<i>EXCSQLSET</i> (on page 377)
	<i>EXCSQLSTT</i> (on page 381)
	<i>INTRDBRQS</i> (on page 445)
	<i>MGRDEPRM</i> (on page 505)
	<i>OBJNSPRM</i> (on page 542)
	<i>OPNQFLRM</i> (on page 554)
	<i>OPNQRY</i> (on page 555)
	<i>PKGBNARM</i> (on page 585)
	<i>PKGBPARM</i> (on page 586)
	<i>PKGNAME</i> (on page 594)
	<i>PKGNAMECSN</i> (on page 596)
	<i>PKGNAMECT</i> (on page 598)
	<i>PRCCNVCD</i> (on page 621)
	<i>PRCCNVRM</i> (on page 625)
	<i>PRMNSPRM</i> (on page 633)
	<i>PRPSQLSTT</i> (on page 636)
	<i>QRYNOPRM</i> (on page 690)
	<i>QRYPOPRM</i> (on page 691)
	<i>RDBACCRM</i> (on page 724)
	<i>RDBAFLRM</i> (on page 725)
	<i>RDBATHRM</i> (on page 727)
	<i>RDBCMM</i> (on page 728)
	<i>RDBNACRM</i> (on page 734)
	<i>RDBNFNRM</i> (on page 739)
	<i>RDBRLLBCK</i> (on page 745)
	<i>RDBUPDRM</i> (on page 751)
	<i>REBIND</i> (on page 753)
	<i>RSCLMTRM</i> (on page 778)
	<i>SECCHK</i> (on page 800)
	<i>SQLERRRM</i> (on page 864)
	<i>SYNCLOG</i> (on page 922)
	<i>SYNTAXRM</i> (on page 989)
	<i>TRGNSPRM</i> (on page 1022)
	<i>VALNSPRM</i> (on page 1057)
<b>mgrdepls</b>	<i>SQLAM</i> (on page 847)
<b>Semantic</b>	<i>CLSQRY</i> (on page 165)
	<i>DRPPKG</i> (on page 293)

*DSCRDBTBL* (on page 300)  
*DSCSQLSTT* (on page 304)  
*ENDBND* (on page 336)  
*PRPSQLSTT* (on page 636)  
*SNDPKT* (on page 829)  
*SQL* (on page 835)  
*SRVLST* (on page 878)

**vldattls**      *RDB* (on page 718)



**NAME**

RDBNFNRM — RDB Not Found

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2211'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

RDB Not Found (RDBNFNRM) Reply Message indicates that the target server cannot find the specified relational database.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'2211'	
<b>rdbnam</b>	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code 8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**cmdrpy** ACCRDB (on page 42)

**Semantic** ACCRDB (on page 42)  
ACCSEC (on page 52)

**NAME**

RDBOVR — Relational Database Overview

**DESCRIPTION (Semantic)**

**Dictionary** QDDTTRD

**Length** \*

**Class** HELP

Relational Database Overview (RDBOVR) provides an overview for applications interacting with remote relational databases (RDBs) to:

- Establish a connection with the remote RDB.
- Create or delete application-specific access algorithms (packages) in the RDB.  
These packages include the definitions of application variables used for input and output and SQL statements the application defines.
- Retrieve descriptions of answer set data.
- Execute SQL statements bound in the RDB package.
- Dynamically prepare and execute SQL statements in the RDB.
- Maintain consistent unit of work boundaries (by using commit and rollback facilities) between the application and the RDB.
- Terminate the connection with the RDB.

The DDM model for performing these operations is independent of the hardware architecture, the operating system, and the local RDB interfaces and facilities of either the source system or the target system.

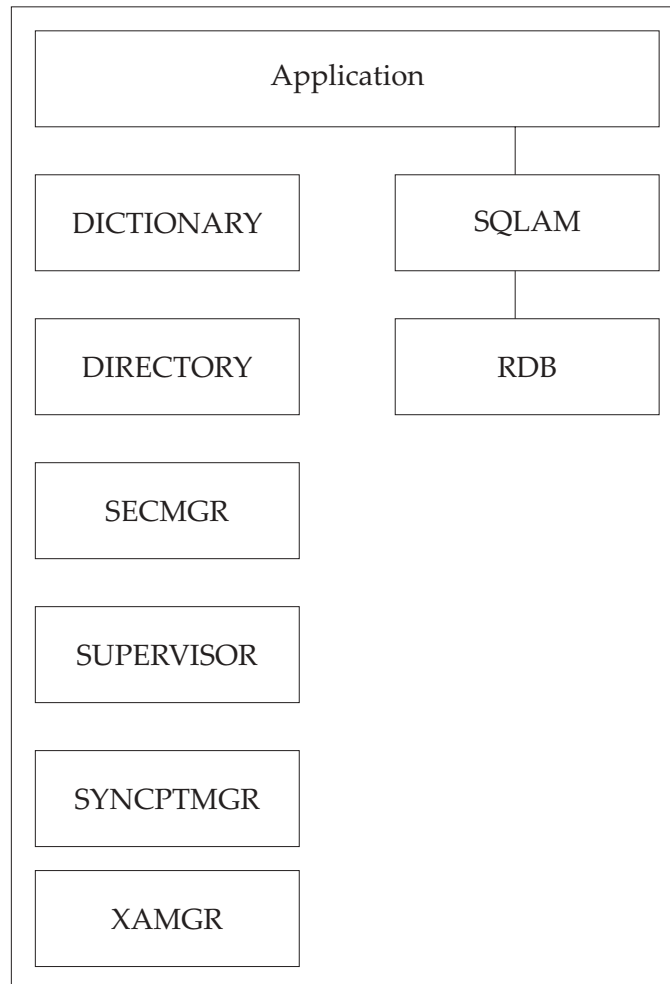
See *SQL* (on page 835) for a discussion of the relationship of SQL and the DDM facilities for distributing SQL statements and data.

**Single-System Model**

DDM defines a single-system model of the relationship between an application and its local RDB. This model is illustrated by Figure 3-68 (on page 741).

In this model, the application uses an instance of an SQLAM and the commands and objects the SQLAM defines to communicate with the RDB. The SQLAM uses other system services (such as directories and the security manager) to establish connectivity with the RDB and to access the RDB. The translation of local system interfaces to or from the command interfaces of the SQLAM is not part of this model.

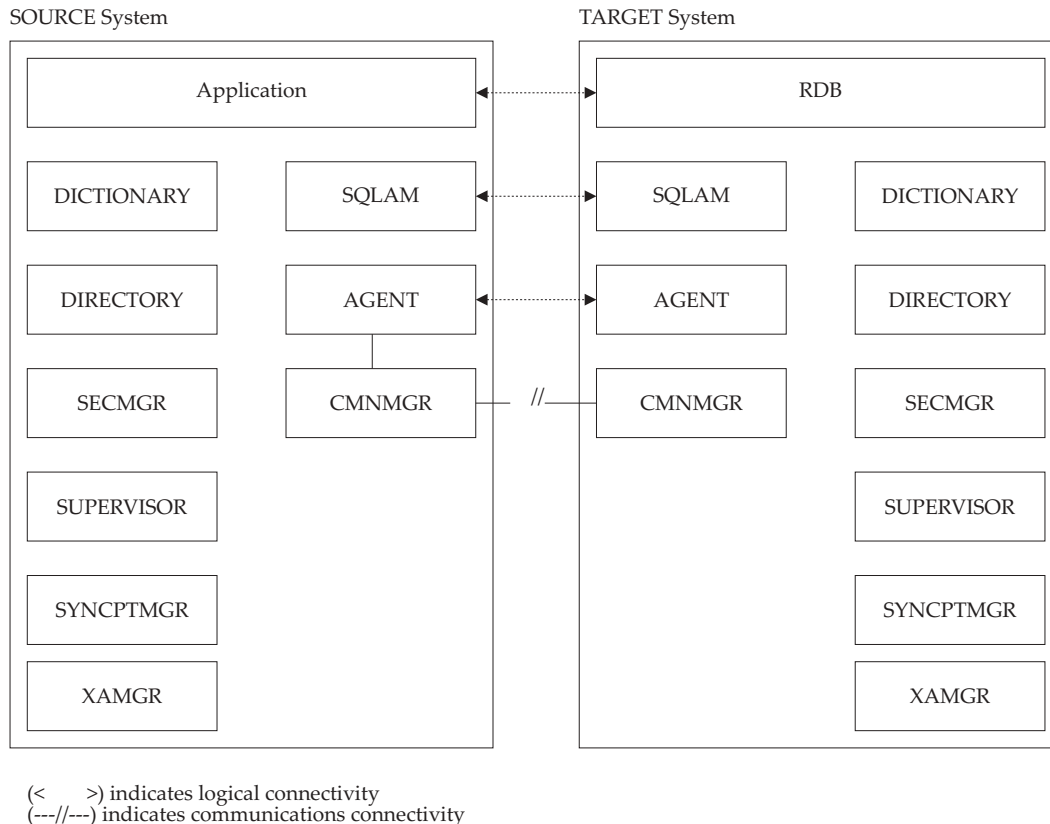
## Single System Model



**Figure 3-68** Application to Local RDB Connection

### Distributed Systems Model

The DDM architecture extends the single-system model in Figure 3-68 to the distributed systems model in Figure 3-69 (on page 742) by allowing the processing of the SQLAM to be split between the source and target systems. DDM agent and communications manager components that are functionally split between the source and target systems provide the communications between the source and target SQLAMs. The source system and the target system can be of different system types. As with other DDM models, individual servers can be implemented according to whatever design best suits their system environment.



**Figure 3-69** Application to Remote RDB Connection

On the source system, the DDM support for RDB access consists of:

1. A source application that obtains RDB services by sending DDM commands to a source SQLAM.

Concerns that the DDM architecture does not address:

- a. The instantiation of an SQLAM for the application to use
  - b. How an SQLAM is instantiated for the application to use
  - c. The mechanisms used to transmit messages between the application and the SQLAM
  - d. The translation of native source system interfaces to DDM commands and objects
  - e. The translation of DDM reply messages and objects to native source system interfaces
2. The source SQLAM performs the following services:
    - a. The SQLAM binds to the source AGENT and sends the ACCRDB command to that agent for transmission to the target agent for processing. This results in the creation of a target SQLAM which binds to the remote RDB.
    - b. In general, the source SQLAM routes DDM commands the application sends to the target SQLAM for processing. However, the source and target SQLAMs cooperate in the processing of many commands. For example, during query processing, both the source and target SQLAMs buffer the data to improve communications facilities.

- c. The source SQLAM converts data between the application and the RDB data representations for all types of SQL data.
  - d. When required, the source SQLAM converts data from the target RDB to the application data representations for all instances of SQL data types.
  - e. When required, the source SQLAM interacts with the source sync point manager (described in the term SYNCPTMGR) for two-phase commit or XA Manager (XAMGR) and backout processing.
3. The source agent (described in the term AGENT) performs the following services:
- a. Routes commands and command data received from the source SQLAM to a source communications manager (described in the term CMNMGR)
  - b. Parses and validates reply messages and reply data received from the source communications manager and routes them to the source SQLAM
  - c. Uses the services of the source system supervisor (described in the term SUPERVISOR), security manager (described in the term SECMGR), and dictionaries (described in the term DICTIONARY)
4. The source communications manager (SCM) manages the communications facilities which support interactions between the source agent and target agent.

It builds a data stream (described in the terms DSS, RQSDSS, and OBJDSS) from the reply messages and reply data resulting from command processing, then transmits this data stream to the source system.

It also extracts reply messages and reply objects from the reply data streams (described in the terms DSS, RPYDSS, and OBJDSS) it receives from the target system for presentation to the source agent. When required, the SCM interacts with the source SYNCPTMGR for two-phase commit and backout processing.

On the target system, DDM support for RDB access consists of:

1. The target communications manager (described in the term CMNMGR)
 

It receives request data streams from the source system, extracts their commands and command data, and then passes them to the target agent for further processing. It builds a data stream from the reply messages and reply data resulting from command processing and transmits it to the source system. When required, the target communications manager (TCM) interacts with the target SYNCPTMGR for two-phase commit and backout processing.
2. The target agent (described in the term AGENT)
 

It receives commands and command data from the TCM, validates them, and routes them to the target manager the command specifies. The agent uses the services of the target system supervisor (described in the term SUPERVISOR), security manager (described in the term SECMGR), directories (described in the term DIRECTORY), and dictionaries (described in the term DICTIONARY). For commands designated to a named RDB, it also routes commands to a target SQLAM.
3. The target SQLAM
 

It manages all accesses to the RDB with a single remote requester. The target SQLAM is bound between the agent and the RDB. Then the bound RDB sends the command to the SQLAM to request processing. When required, the target SQLAM interacts with the target SYNCPTMGR for two-phase commit or XAMGR and backout processing.

## 4. A target server

It can support zero or more named RDBs (described in the term RDB).

All operations the RDB performs occur within a unit of work. A unit of work is implicitly started as soon as an application starts a transaction or by the completion of the previous unit of work.

A unit of work consists of zero or more requests for RDB operations, and it is either committed or rolled back. If the unit of work is committed, all changes made to the RDB within the unit of work are permanent and revealed to other requesters. If the unit of work is rolled back, all changes to the RDB within the unit of work are removed.

When using the CMNSYNCP, a SYNCPTMGR is at the source and target coordinates the commit or rollback processing.

Application termination (or some other unarchitected condition) determines the end of remote RDB access. The SQL application at the source system should explicitly use the SQL commit or rollback statements prior to termination. If the SQL application terminates normally without explicit commit or rollback, a commit function is issued for the application before the conversation is terminated. If the SQL application terminates abnormally, then a rollback function is issued for the application before terminating the conversation. The DDM managers used during access are released, along with the communications resources that were used.

**SEE ALSO**

**Semantic**      *MGROVR* (on page 514)  
                  *RDB* (on page 718)  
                  *SQL* (on page 835)  
                  *SQLAM* (on page 847)

**NAME**

RDBRLLBCK — RDB Rollback Unit of Work

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'200F'**Length** \***Class** CLASS**Sprcls** COMMAND - Command

RDB Rollback Unit of Work (RDBRLLBCK) Command rolls back all work performed for the current unit of work. The current unit of work ends, and a new unit of work begins.

**Source System Processing**

The source system determines the location of the RDB:

Local            Call the local RDB server.

Remote          Send the RDBRLLBCK command to the remote RDB server.

If the connection is to be reused at the completion of the commit or rollback, the release connection parameter (*rlsconv*) is set to REUSE. The connection cannot be reused until the receipt of the ENDUOWRM. If the target server responds with *rlsconv* set to FALSE, the connection is not allowed to be reused until the application terminates. If set to REUSE, the target server returns a set of SQLSTT objects to establish the application execution environment.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**DSS Carrier: RQSDSS****Target System Processing**

The *rdbnam* parameter specifies the name of the RDB to which the ACCRDB command gained access. If the *rdbnam* parameter is specified, then its value must be the same as the value specified for the *rdbnam* parameter on the ACCRDB command.

If an RDBRLLBCK occurs after a BGNBND command is issued and before the package binding process is terminated, then the binding process is implicitly terminated before the unit of work is rolled back.

If an RDBRLLBCK occurs after an OPNQRY command is issued and before the opened query is terminated, then the query is implicitly closed before the unit of work is rolled back.

Normal completion of this command results in an ENDUOWRM sent prior to any reply data object.

The TYPDEFNAM reply data object specifies the name of the data type to data representation mapping definitions that the target SQLAM uses when sending the SQLCARD object for this command.

The TYPDEFOVR reply data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type to data representation mapping definitions for the SQLCARD object for this command.

### Data Conversion

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

### Exceptions

Exception conditions that the RDB detects are reported in an SQLCARD object.

If access to the specified RDB is currently not possible, then the command is rejected with the RDBNACRM.

If the CMNSYNCPT is used, then the RDBRLLBCK command is rejected with the CMDVLTRM.

### Source System Reply Processing

Return any reply messages and data objects to the requester.

### Data Conversion

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'200F'	
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	
<b>cmddta</b>	NIL	
rlsconv	INSTANCE_OF ENUVAL ENUVAL NOTE  DFTVAL OPTIONAL MINLVL	RLSCONV - Release Connection X'F0'(NO) - Do not reuse connection X'F2'(REUSE) - Connection can be reused X'F1'(TERMINATE) is not allowed. If specified, it is a conversation protocol error.  X'F0'(NO) - Do not reuse connection  7
<b>rpydta</b>	SQLSTT	REPLY OBJECTS
X'002F'	INSTANCE_OF	TYPDEFNAM - Data Type Definition Name



	ENUVAL	'QTDSQL370'
	ENUVAL	'QTDSQL400'
	ENUVAL	'QTDSQLX86'
	ENUVAL	'QTDSQLASC'
	MINLVL	4
	ENUVAL	'QTDSQLVAX'
	MINLVL	4
	DFTVAL	''
	NOTE	The default means the value received on ACCRDBRM is used.
OPTIONAL		
X'0035'	INSTANCE_OF	TYPDEFOVR - TYPDEF Overrides
	OPTIONAL	
	DFTVAL	''
	NOTE	The default means the value received on ACCRDBRM is used.
X'1C03'	INSTANCE_OF	MGRVLVLOVR - Manager Level Overrides
	OPTIONAL	
	MINLVL	7
X'2408'	INSTANCE_OF	SQLCARD - SQL Communications
	REQUIRED	Area Reply Data
<b>cmdrpy</b>		
<b>COMMAND REPLIES</b>		
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported
X'221D'	INSTANCE_OF	CMDVLTRM - Command Violation
	MINLVL	4
X'220C'	INSTANCE_OF	ENDUOWRM - End Unit of Work Condition
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'2204'	INSTANCE_OF	RDBNACRM - RDB Not Accessed
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limits Reached
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'125F'	INSTANCE_OF	TRGNSPRM - Target Not Supported
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

**SEE ALSO**

<b>inscmd</b>	<i>SQLAM</i> (on page 847)
<b>Semantic</b>	<i>CMDVLTRM</i> (on page 181)
	<i>INTRDBRQS</i> (on page 445)
	<i>OPNQRY</i> (on page 555)
	<i>SQL</i> (on page 835)
	<i>SQLAM</i> (on page 847)
	<i>SUBSETS</i> (on page 902)
	<i>XAMGROV</i> (on page 1066)

## NAME

RDBRLSCMM — RDB Release at Commit

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD

**Codepoint** X'2438'

**Length** \*

**Class** codpnt

RDB Release at Commit (RDBRLSCMM) specifies that the RDB releases the package execution resources every time a unit of work is either committed or rolled back.

## SEE ALSO

**insvar** *RDBRLOPT* (on page 750)

**NAME**

RDBRLSCNV — RDB Release at Conversation Deallocation

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2439'**Length** \***Class** codpnt

RDB Release at Conversation Deallocation (RDBRLSCNV) specifies that the target RDB releases the package execution resources when the conversation with the source server is deallocated.

**SEE ALSO****insvar** *RDBRLOPT* (on page 750)

**NAME**

RDBLSOPT — RDB Release Option

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2129'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

RDB Release Option (RDBLSOPT) String specifies when the RDB releases the package execution resources and the associated serialization or sharing locks.

The RDB allocates a set of resources for executing package SQL statements or for executing a specific package SQL statement. These resources include, but are not limited to, the physical files containing RDB objects (such as a table) and the serialization or sharing intent locking on the physical files.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	6	
<b>class</b>	X'2129'	
<b>value</b>	INSTANCE_OF	CODPNTDR - Codepoint Data Representation
	ENUVAL	X'2438' - RDBRLSCMM - RDB Release at Commit
	ENUVAL	X'2439' - RDBRLSCNV - RDB Release at Conversation Deallocation
	DFTVAL	X'2438' - RDBRLSCMM - RDB Release at Commit
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *BGNBND* (on page 110)  
*REBIND* (on page 753)

**NAME**

RDBUPDRM — RDB Update Reply Message

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2218'  
**Length** \*  
**Class** CLASS  
**Sprcls** RPYMSG - Reply Message

RDB Update Reply Message (RDBUPDRM) indicates that a DDM command resulted in an update at the target relational database (RDB). If a command generates multiple reply messages including RDBUPDRM, then the RDBUPDRM must be the first reply message returned for the command.

For each target server, the RDBUPDRM must be returned the first time an update is made to the target RDB within a unit of work (UOW). The target server may optionally return the RDBUPDRM after subsequent updates within the UOW. If multiple target RDBs are involved with the current UOW and updates are made with any of them, then the RDBUPDRM must be returned in response to the first update at each of them.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'2218'	
<b>rdbnam</b>	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF ENUVAL REQUIRED	SVRCOD - Severity Code 0 - INFO - Information Only Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**cmdrpy** *BGNBND* (on page 110)  
*BNDSQLSTT* (on page 136)  
*CNTQRY* (on page 222)  
*DRPPKG* (on page 293)  
*ENDATMCHN* (on page 333)  
*ENDBND* (on page 336)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSTT* (on page 381)  
*OPNQRY* (on page 555)  
*REBIND* (on page 753)

**Semantic**

*BGNBND* (on page 110)  
*BNDSQLSTT* (on page 136)  
*CNTQRY* (on page 222)  
*DRPPKG* (on page 293)  
*ENDATMCHN* (on page 333)  
*ENDBND* (on page 336)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSTT* (on page 381)  
*ISOLVLNC* (on page 455)  
*OPNQRY* (on page 555)  
*REBIND* (on page 753)  
*SYNCPTOV* (on page 944)

**NAME**

REBIND — Rebind an Existing RDB Package

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2010'

**Length** \*

**Class** CLASS

**Sprcls** COMMAND - Command

Rebind an Existing RDB Package (REBIND) Command rebinds an existing relational database Package.

The function of the REBIND command is equivalent to the following sequence of commands:

1. A BGNBND command with PKGRPLOPT(PKGRPLALW) specified
2. A BNDSQLSTT command for each SQL statement that exists in the package
3. An ENDBND command

**Source System Processing**

The source system determines the location of the RDB:

Local Call the local RDB server.

Remote Send the REBIND command to the remote RDB server.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**DSS Carrier: RQSDSS****Target System Processing**

The *bndchkexs* parameter controls whether the lack of a named RDB object or authorization of a requester to a named RDB object is treated as an error during the bind (rebind) process.

The *bndexopt* parameter controls whether the target SQLAM causes the target RDB to explain all explainable SQL statements bound into the package.

The *dftrdbcol* parameter specifies the default RDB collection identifier that the target RDB uses to perform, if necessary, RDB object name completion functions on SQL statements bound into the package.

The *dgriopl* parameter specifies the degree of I/O parallel processing static SQL queries bound into the package use if the RDB supports the I/O parallel processing.

The *pkgathrul* parameter specifies which authorization identifier to use when dynamic SQL in a package is executed. The possible alternatives are either the requester (the person executing the package) or the owner (the person who owns the package).

The *pkgisolvl* parameter specifies the isolation level that the RDB uses when executing SQL statements in this package unless a target RDB runtime mechanism overrides it.

The *pkgnam* parameter specifies the fully qualified name of the existing package which is rebinding.

The *pkgownid* parameter specifies the end-user name (identifier) of the the package's owner.

The *rdbnam* parameter specifies the name of the RDB to which the ACCRDB command gained access. This is the RDB that the existing package is being rebound to.

The *rdbrlsopt* parameter specifies when the RDB releases the package execution resources for this package.

The *vrnam* parameter specifies the version name associated with the named existing package.

Some items of the existing package cannot be changed during the rebind process. These are:

- The fully qualified package name (PKGNAME)
- The package consistency token (PKGCNSTKN)
- The package version name (VRSNAME)
- The statement delimiter for delimited statement strings and identifiers (STTSTRDEL)
- The statement decimal delimiter (STTDECDEL)
- The statement date format (STTDATFMT)
- The statement time format (STTTIMFMT)
- The package default character subtype for SQL character columns (PKGDFTCST)
- The package default CCSIDs for character and graphic columns (PKGDFTCC)
- The query blocking control (QRYBLKCTL)
- The package title (TITLE)

If the *pkgownid* parameter is specified or if the requester is different than the current package owner, then the REBIND command requires the same authorizations as a BGNBND command that specified PKGATHOPT(PKGATHOPT) and PKGRPLOPT(PKGRPLALW).

Additional bind options can be sent in occurrences of BNDOPT cmd data objects. If the target server cannot recognize or process any of the bind options then it responds with a non-zero SQLSTATE in the SQLCARD. The SQLERRMC error field in the SQLCARD indicates the bind option in error. If there are multiple bind options in error, only the first one is reported. The target server has the option of terminating the bind process with an error SQLSTATE or continuing by sending a warning SQLSTATE.

The TYPDEFNAM reply data object specifies the name of the data type to data representation mapping definition that the target SQLAM uses when sending the SQLCARD object for this command.

The TYPDEFOVR reply data object specifies the single-byte, double-byte, and mixed-byte CCSIDs of the SDAs in the identified data type to data representation mapping definitions for the SQLCARD object for this command. If the REBIND command completes successfully, then an SQLCARD object is returned. An RDBUPDRM must be returned if the command makes recoverable updates in the RDB and if the server did not return a prior RDBUPDRM in the current unit of work. A recoverable update is an RDB update that writes information to the recovery log of the RDB.



**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**Exceptions**

Exception conditions that the RDB detects are reported in an SQLCARD object.

If a REBIND command is issued before a previous bind process (BGNBND or REBIND command) has been terminated, then the command is rejected with the PKGBPARM.

If access to the specified RDB is not current, then the command is rejected with the RDBNACRM.

**Source System Reply Processing**

Return any reply messages and data objects to the requester.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
<b>length</b>	*	
<b>class</b>	X'2010'	
<b>bndchkexs</b>	INSTANCE_OF OPTIONAL DFTVAL NOTE	BNDCHKEXS - Bind Existence Checking " Defaults to the value specified for the existing package.
<b>bndexpopt</b>	INSTANCE_OF OPTIONAL DFTVAL NOTE	BNDEXPOPT - Bind Explain Option " Defaults to the value specified for the existing package.
<b>dftrdbcol</b>	INSTANCE_OF OPTIONAL DFTVAL NOTE	DFTRDBCOL - Default RDB Collection ID " Defaults to the value specified for the existing package.
<b>dgrioprl</b>	INSTANCE_OF OPTIONAL DFTVAL NOTE	DGRIOPRL - Degree of I/O Parallelism " If this parameter is not specified, then I/O parallel processing occurs to the degree currently in effect for the bound package.
	IGNORABLE	

	MINLVL	4
pkgathrul	INSTANCE_OF OPTIONAL DFTVAL NOTE IGNORABLE MINLVL	PKGATHRUL - Package Authorization Rules " Defaults to the value specified for the existing package. 5
pkgisolvl	INSTANCE_OF OPTIONAL DFTVAL NOTE	PKGISOLVL - Package Isolation Level " Defaults to the value specified for the existing package.
pkgnam	INSTANCE_OF REQUIRED	PKGNAME - RDB Package Name
pkgownid	INSTANCE_OF OPTIONAL DFTVAL NOTE	PKGOWNID - Package Owner Identifier " Defaults to the value specified for the existing package.
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
rdbrlsopt	INSTANCE_OF OPTIONAL DFTVAL NOTE	RDBRLOPT - RDB Release Option " Defaults to the value specified for the existing package.
vrsnam	INSTANCE_OF OPTIONAL	VRSNAM - Version Name
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>cmddta</b>		<b>COMMAND OBJECTS</b>
X'2405'	INSTANCE_OF OPTIONAL REPEATABLE MINLVL	BNDOPT - Bind Option 5
<b>rpydta</b>		<b>REPLY OBJECTS</b>
X'002F'	INSTANCE_OF ENUVAL ENUVAL ENUVAL ENUVAL MINLVL ENUVAL MINLVL OPTIONAL DFTVAL NOTE	TYPDEFNAM - Data Type Definition Name 'QTDSQL370' 'QTDSQL400' 'QTDSQLX86' 'QTDSQLASC' 4 'QTDSQLVAX' 4 " The default means the value received on the ACCRDBRM is used.

X'0035'	INSTANCE_OF OPTIONAL DFTVAL NOTE	TYPDEFOVR - TYPDEF Overrides " The default means the value received on the ACCRDBRM is used.
X'1C03'	INSTANCE_OF OPTIONAL MINLVL	MGRVLVLOVR - Manager Level Overrides 7
X'2408'	INSTANCE_OF REQUIRED	SQLCARD - SQL Communications Area Reply Data
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'220D'	INSTANCE_OF	ABNUOWRM - Abnormal End Unit of Work Condition
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported
X'1253'	INSTANCE_OF	OBJNSPRM - Object Not Supported
X'2209'	INSTANCE_OF	PKGBPARAM - RDB Package Binding Process Active
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'2204'	INSTANCE_OF	RDBNACRM - RDB Not Accessed
X'2218'	INSTANCE_OF MINLVL	RDBUPDRM - RDB Update Reply Message 4
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limits Reached
X'2213'	INSTANCE_OF	SQLERRRM - SQL Error Condition
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'125F'	INSTANCE_OF	TRGNSPRM - Target Not Supported
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

**SEE ALSO**

<b>inscmd</b>	<i>SQLAM</i> (on page 847)
<b>Semantic</b>	<i>BGNBND</i> (on page 110) <i>BNDCHKEXS</i> (on page 125) <i>BNDEXSQR</i> (on page 131) <i>SQL</i> (on page 835) <i>SQLAM</i> (on page 847)

**NAME**

REPEATABLE — Repeatable Variable Attribute

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'0031'

**Length** \*

**Class** CLASS

**Sprcls** OBJECT - Self-identifying Data

Repeatable Variable Attribute (REPEATABLE) specifies that a parameter, value, or object can be repeated in the value of the object variable being described. There are no requirements that:

- The elements of the list be unique
- The elements of the list be in any order

If a list of enumerated values is also specified, more than one of the enumerated values can be specified.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
<b>length</b>	*
<b>class</b>	X'0031'
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO**

**cmdtda** *BGNBND* (on page 110)  
*BNDSQLSTT* (on page 136)  
*EXCSQLSET* (on page 377)  
*REBIND* (on page 753)  
*SYNCCTL* (on page 915)

**insvar** *ARRAY* (on page 101)  
*ATTLST* (on page 105)  
*BITSTRDR* (on page 124)  
*BYTSTRDR* (on page 146)  
*CHRSTRDR* (on page 156)  
*DCTIND* (on page 258)  
*DEFLST* (on page 268)  
*FDOOBJ* (on page 411)  
*HEXSTRDR* (on page 429)  
*MGRLVLLS* (on page 508)  
*NAMSYMDR* (on page 528)  
*PKGSNLST* (on page 607)  
*SECMEC* (on page 811)  
*SQLDTA* (on page 860)  
*SQLDTARD* (on page 862)  
*SRVLSRV* (on page 876)  
*SRVLST* (on page 878)

<b>rpydta</b>	<i>CNTQRY</i> (on page 222) <i>EXCSQLSTT</i> (on page 381) <i>OPNQRY</i> (on page 555)
<b>Semantic</b>	<i>ENUCLS</i> (on page 345)
<b>semantic</b>	<i>CODPNT</i> (on page 233) <i>CONSTANT</i> (on page 244) <i>HELP</i> (on page 425)

**NAME**

REQUESTER — Package Requester Authorization Identifier

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Length** \*

**Class** CONSTANT

Package Requester Authorization Identifier (REQUESTER) value indicates that the authorization identifier used for the execution of a dynamic SQL package is the authorization identifier of the end user (the person requesting execution of the package).

---

value 0

**SEE ALSO**

**insvar** *PKGATHRUL* (on page 581)

**Semantic** *PKGATHRUL* (on page 581)

**NAME**

REQUIRED — Required Value Attribute

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'0032'

**Length** \*

**Class** CLASS

**Sprcls** OBJECT - Self-identifying Data

Required Value Attribute (REQUIRED) specifies that support or use of a class, command, parameter, or value is required within the rules for subsetting listed in *SUBSETS* (on page 902).

1. When specified for a command in the command list of a class, all receivers must support the command if they support the class.
2. When specified for a parameter in the parameter list of a command, the parameter must be sent for that command. All receivers supporting the command must recognize and process the parameter as defined.
3. When specified for a parameter in the parameter list of a reply message, the parameter must be sent for that reply message. All receivers must accept the parameter.
4. When specified for a value in the value list of a parameter, the value must be sent for the parameter. For example, the year, month, and day values of a DATE parameter are required. All receivers supporting the parameter must recognize the value.
5. If specified for a command data object in the command data object list of a command, the command data object must be sent for that command. All receivers supporting the command must recognize and process the command data object as defined.
6. If specified for a reply data object in the reply data object list of a command, the reply data object is normally returned for the command. When exception conditions occur, the reply data object may not be returned, and reply messages return a description of the exception conditions instead.

If REQUIRED is specified for multiple entities that specify each other to be mutually-exclusive (MTLEXC attribute), then only one of the required entities can be specified.

<b>clsvar</b>	NIL
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
<b>length</b>	*
<b>class</b>	X'0032'
<b>clscmd</b>	NIL
<b>inscmd</b>	NIL

**SEE ALSO**

None.

## NAME

RESERVED — Reserved Value Attribute

## DESCRIPTION (Semantic)

**Dictionary** QDDPRMD**Codepoint** X'0033'**Length** \***Class** CLASS**Sprcls** SCALAR - Scalar Object

Reserved Value Attribute (RESERVED) specifies that a value is reserved only for DDM architecture and cannot be used for any other purposes.

The attribute object must have attributes compatible with those specified for the term.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'0033'	
<b>value</b>	SPRCLS NOTE	FIELD - A Discrete Unit of Data The attributes for this variable vary according to what is being defined. Also, when BITDR, BITSTRDR, HEXDR, or HEXSTRDR values are specified, they are represented as being left-justified in the fewest number of whole bytes possible. For example, 123 would be X'1230'.
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

## SEE ALSO

**insvar** *DSSFMT* (on page 318)  
*RSLSETFLG* (on page 783)



**NAME**

RESPKTSZ — Response Packet Size

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1908'

**Length** 4

**Class** CLASS

**Sprcls** BIN - Binary Integer Number

Response Packet Size (RESPKTSZ) specifies the size of the PKTOBJ object that a target server must return to a source server in response to a SNDPKT command.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	8	
class	X'1908'	
value	INSTANCE_OF	BINDR - Binary Number Field
	MINLVL	7
	LENGTH	32
	MINVAL	0
	DFTVAL	1000
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *SNDPKT* (on page 829)

**NAME**

RESYNQVR — Resynchronization Overview

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

Resynchronization Overview (RESYNQVR) discusses the resynchronization process.

The following information on resynchronization processing has been extracted from the *SNA LU 6.2 Reference: Peer Protocols* (SC31-6808, IBM).

Resynchronization (resync) occurs if two-phase commit processing is interrupted by a resource failure. A resource failure can be caused by a node failure, a session failure, a program failure, or other problem detected by a protected resource manager. The resource failure might be between the sync point services and local resource managers or sync point services and remote resource managers.

Resynchronization (resync) is conducted independently for each failed protected resource for which it is required. Resync has the following purposes:

- To place distributed resources in consistent states; if this is not possible, to notify the operators at the logical unit that detected the damage and at the unit of the root of the sync point tree
- To unlock locked resources in order to free them for other uses
- To update the log showing that no more sync point work is needed for that protected resource for that unit of work

Resync for a conversation resource occurs even though further work is impossible on the associated protected conversation due to a conversation failure or to the transaction programs terminating. The SYNCPTMGR in one partner logical unit establishes a special conversation with a resynchronization transaction program in the partner logical unit to resynchronize the unit of work state for each failed conversation which resync is required. The protected resources are thereby left in consistent states if possible, even though further work is impossible without establishing new conversations and thus, invoking a new version of the transaction program on at least one side.

**SEE ALSO**

**Semantic**            *CMNSYNCPT* (on page 202)  
                           *SYNCMNFL* (on page 928)

**NAME**

RLSCONV — Release Conversation

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'119F'

**Length** \*

**Class** CLASS

**Sprcls** HEXSTRDR - Hexadecimal String

Release Conversation (RLSCONV) specifies whether a conversation should be terminated or reused at the completion of the sync point operation. If the commit decision is to commit, the conversation is terminated. If the commit decision is to rollback, the conversation is not terminated. If reuse is requested the target server returns the same value to indicate whether the connection can be reused by another application.

For XAMGR protected connections, the RLSCONV can be sent on any operation that does not associate the connection with an XID. XAMGR protected connections can be pooled to a different application, as long as it is not associated with any XID. If the application server cannot dissociate an XID from the connection, than it must always reply with RLSCONV(NO).

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	5	
class	X'119F'	
value	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	LENGTH_OF	2
	ENUVAL	X'F2' (REUSE) - Reuse released connection
	ENUVAL	X'F1' (TERMINATE) - Terminate released connection
	ENUVAL	X'F0' (NO) - Do not terminate or reuse connection
	DFTVAL	X'F0' (NO) - Do not terminate or reuse connection
	MINLVL	7
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *SYNCCTL* (on page 915)

**inscmd** *RDBCMM* (on page 728)

**Semantic** *SYNCCRD* (on page 913)  
*XAMGROV* (on page 1066)

NAME

RPYDSS — Reply Data Stream Structure

DESCRIPTION (Semantic)

**Dictionary** QDDBASD  
**Codepoint** X'1436'  
**Length** \*  
**Class** CLASS  
**Sprcls** DSS - Data Stream Structures

Reply Data Stream Structure (RPYDSS) specifies the general format of the DDM reply data stream structures. All fields must be specified in the order shown in Figure 3-70 and in the *insvar* section because they are not self-defining structures.

An RPYDSS can carry multiple reply messages if all of the reply messages are for the same correlation identifier. Multiple RPYDSSs can be chained for transmission. When the DDM conversational protocol (CMNAPPC, CMNSYNCPT, or CMNTCPIP) is used, the object structures (OBJDSSs) associated with a command must be chained with the RPYDSSs associated with that command.

Figure 3-70 maps a reply message onto a data stream.

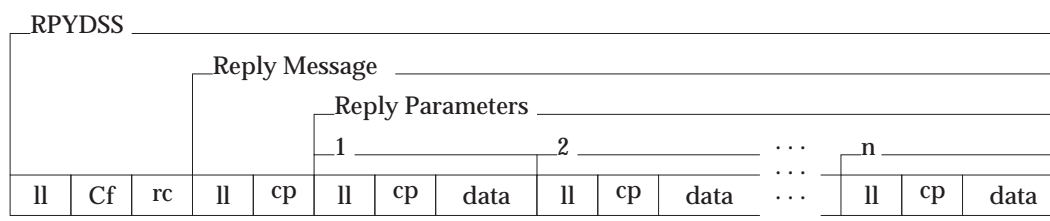


Figure 3-70 Reply Data Stream Structures

Legend

- ll** Two-byte length field
- C** D0
- f** One-byte format (request, reply, object, or communications structure)
- rc** Two-byte request correlation identifier
- cp** Codepoint of an object
- data** Value of an object

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	INSTANCE_OF	BINDR - Binary Number Field

	LENGTH	16
	MINVAL	6
	MAXVAL	32,767
	NOTE	Specifies the length of a reply structure. If it is necessary to include more data than the maximum allows, or if it is desirable to break a large structure into smaller sections, set the high-order bit of the length field to 1. This indicates that the structure is continued in the next structure transmitted.
	REQUIRED	
ddmid	INSTANCE_OF	DDMID - DDM Identifier
	REQUIRED	
format	INSTANCE_OF	DSSFMT - Data Stream Structure Format
	ENUVAL	X'02'
	NOTE	A reply message.
	ENUVAL	X'42'
	NOTE	A reply message, chained: next DSS has a different request correlator (RQSCRR).
	ENUVAL	X'52'
	NOTE	A reply message, chained: next DSS has the same request correlator (RQSCRR).
	REQUIRED	
rqscrr	INSTANCE_OF	RQSCRR - Request Correlation Identifier
	NOTE	Specifies a request identifier the source communications manager assigns. This identifier is the same for all DSSs containing a request or containing data associated with a request the source communications manager is sending. The same identifier is also specified for all DSSs containing replies to or data associated with that request.
	REQUIRED	
data	MINLEN	0
	MAXLEN	32761
	NOTE	Instances of a defined class of reply messages are mapped into this data field. The codepoint of the message identifies a specific reply message (for instance 120B identifies the ENDFILRM).
	REQUIRED	
clscmd	NIL	
inscmd	NIL	

## SEE ALSO

<b>insvar</b>	<i>OBJDSS</i> (on page 536) <i>PRCCNVCD</i> (on page 621)
<b>Semantic</b>	<i>ACCRDBRM</i> (on page 48) <i>AGNRPYPR</i> (on page 67) <i>APPCMNFL</i> (on page 68) <i>APPCMNI</i> (on page 72) <i>APPSRCCD</i> (on page 79) <i>APPSRCCR</i> (on page 86) <i>APPSRCER</i> (on page 91) <i>APPTRGER</i> (on page 95) <i>BGNBNDRM</i> (on page 117) <i>CMDATHRM</i> (on page 171) <i>CMDCHKRM</i> (on page 173) <i>CMDCMPRM</i> (on page 175) <i>CMDNSPRM</i> (on page 176) <i>CMDVLTRM</i> (on page 181) <i>CMMRQSRM</i> (on page 182) <i>CMNAPPC</i> (on page 184) <i>CMNMGR</i> (on page 196) <i>CMNSYNCPT</i> (on page 202) <i>CMNTCPIP</i> (on page 214) <i>COMMAND</i> (on page 240) <i>DSCINVRM</i> (on page 298) <i>DSS</i> (on page 308) <i>DTAMCHRM</i> (on page 320) <i>ENDQRYRM</i> (on page 342) <i>ENDUOWRM</i> (on page 343) <i>INHERITANCE</i> (on page 437) <i>INTTKNRM</i> (on page 448) <i>LMTBLKPRC</i> (on page 475) <i>MGRDEPRM</i> (on page 505) <i>MGRLVLRM</i> (on page 512) <i>OBJNSPRM</i> (on page 542) <i>OPNQFLRM</i> (on page 554) <i>OPNQRYRM</i> (on page 566) <i>PKGBNARM</i> (on page 585) <i>PKGBPARM</i> (on page 586) <i>PRCCNVRM</i> (on page 625) <i>PRMNSPRM</i> (on page 633) <i>QRYBLK</i> (on page 670) <i>QRYNOPRM</i> (on page 690) <i>QRYPOPRM</i> (on page 691) <i>RDBACCRM</i> (on page 724) <i>RDBAFLRM</i> (on page 725) <i>RDBATHRM</i> (on page 727) <i>RDBNACRM</i> (on page 734) <i>RDBNFNRM</i> (on page 739) <i>RDBUPDRM</i> (on page 751) <i>RPYMSG</i> (on page 770) <i>RQSCR</i> (on page 772)

*RSCLMTRM* (on page 778)  
*RSLSETRM* (on page 785)  
*SECCHKRM* (on page 809)  
*SQLERRRM* (on page 864)  
*SYNCMNFL* (on page 928)  
*SYNCMNI* (on page 931)  
*SYNTAXRM* (on page 989)  
*TCPCMNFL* (on page 992)  
*TCPCMNI* (on page 994)  
*TCPSRCCD* (on page 1007)  
*TCPSRCER* (on page 1013)  
*TRGNSPRM* (on page 1022)  
*VALNSPRM* (on page 1057)

**NAME**

RPYMSG — Reply Message

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1437'

**Length** \*

**Class** CLASS

**Sprcls** COLLECTION - Collection Object

Reply Message (RPYMSG) is a message returned to the sender about a condition that occurred during the processing of a command. A single command can generate several reply messages.

All reply messages contain a severity code parameter that characterizes the severity of the condition reported. Specific reply messages are defined as subclasses of RPYMSG and define specific additional parameters to be returned with the message.

Before transmission, the DDM communications manager maps all RPYMSG objects onto the RPYDSS.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'1437'	
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
svrcod	INSTANCE_OF REQUIRED	SVRCOD - Severity Code
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

- Semantic**
- ABNUOWRM* (on page 39)
  - ACCRDBRM* (on page 48)
  - AGNPRMRM* (on page 65)
  - BGNBNDRM* (on page 117)
  - CMDATHRM* (on page 171)
  - CMDCHKRM* (on page 173)
  - CMDCMPRM* (on page 175)
  - CMDNSPRM* (on page 176)
  - CMDVLTRM* (on page 181)
  - CMMRQSRM* (on page 182)
  - DSCINVRM* (on page 298)
  - DSS* (on page 308)
  - DTAMCHRM* (on page 320)
  - ENDQRYRM* (on page 342)
  - ENDUOWRM* (on page 343)
  - INHERITANCE* (on page 437)
  - INTTKNRM* (on page 448)
  - MGRDEPRM* (on page 505)



*MGRLVLRM* (on page 512)  
*OBJNSPRM* (on page 542)  
*OPNQFLRM* (on page 554)  
*OPNQRYRM* (on page 566)  
*PKGBNARM* (on page 585)  
*PKGBPARM* (on page 586)  
*PRCCNVRM* (on page 625)  
*PRMNSPRM* (on page 633)  
*QRYNOPRM* (on page 690)  
*QRYPOPRM* (on page 691)  
*RDBACCRM* (on page 724)  
*RDBAFLRM* (on page 725)  
*RDBATHRM* (on page 727)  
*RDBNACRM* (on page 734)  
*RDBNFNRM* (on page 739)  
*RDBUPDRM* (on page 751)  
*RSCLMTRM* (on page 778)  
*RSLSETRM* (on page 785)  
*SECCHKRM* (on page 809)  
*SQLERRRM* (on page 864)  
*SYNTAXRM* (on page 989)  
*TRGNSPRM* (on page 1022)  
*VALNSPRM* (on page 1057)

**NAME**

RQSCRR — Request Correlation Identifier

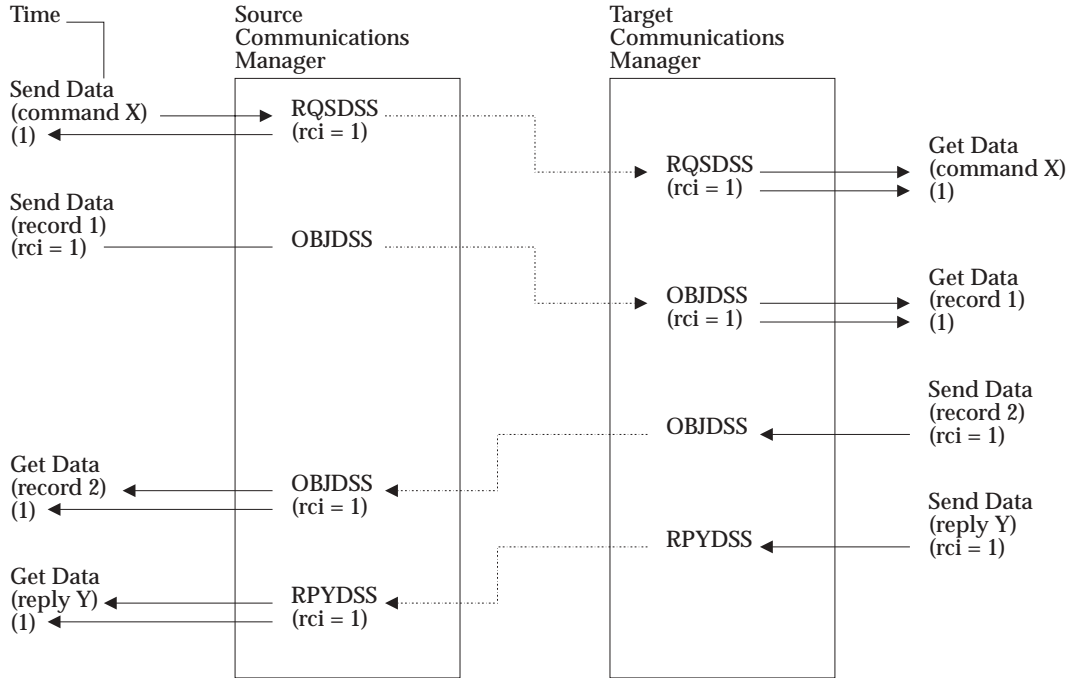
**DESCRIPTION (Semantic)****Dictionary** QDDBASD**Codepoint** X'1438'**Length** \***Class** CLASS**Sprcls** DATA - Encoded Information

Request Correlation Identifier (RQSCRR) generated by the source communications manager ties together:

- A request
- Its request data
- Replies to the request
- Data returned to the requester

The request correlator can be set to any positive (number greater than zero) binary number for the first request, or only request, in an RQSDSS chain. Each RQSDSS in an RQSDSS chain after the first one must have a request correlator that is greater than the preceding RQSDSS. The request correlator can be set to any value.

Figure 3-71 (on page 773) illustrates the request correlation for a single command and its data and replies. Each RQSDSS in a chain must have a unique correlation identifier.



rci = two-byte request correlation identifier.

**Figure 3-71** Data Stream Structure Request Correlation

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
<b>value</b>	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	16
	SPCVAL	-1
	NOTE	Specify the special value of minus one whenever the request correlator of the data stream structure being processed is indeterminable.
	MINVAL	1
	REQUIRED	
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

- insvar**
  - OBJDSS* (on page 536)
  - RPYDSS* (on page 766)
  - RQSDSS* (on page 774)
- Semantic**
  - AGNRPYPR* (on page 67)
  - DSS* (on page 308)
  - SUBSETS* (on page 902)

**NAME**

RQSDSS — Request Data Stream Structure

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1439'

**Length** \*

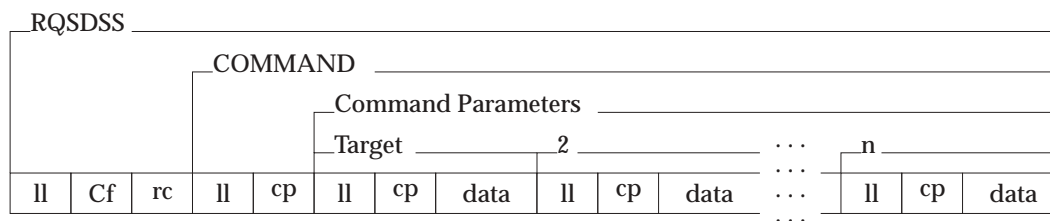
**Class** CLASS

**Sprcls** DSS - Data Stream Structures

Request Data Stream Structure (RQSDSS) specifies the general format of request data stream structures. All RQSDSS fields must be specified in the order shown in Figure 3-72 and in the *insvar* section because they are not self-defining structures.

An RQSDSS can carry only one command, but multiple RQSDSSs can be chained for transmission and sequential execution. When the DDM conversational protocol (CMNAPPC, CMNSYNCPT, or CMNTCPIP) is used, the object structures (OBJDSSs) associated with a command must be chained following that RQSDSS carrying the command.

Figure 3-72 maps a command onto the data stream.



**Figure 3-72** Request Data Stream Structures

**Legend**

- ll** Two-byte length field
- C** D0
- f** One-byte format (request, reply, object, or communications structure)
- rc** Two-byte request correlation identifier
- cp** Codepoint of an object
- data** Value of an object

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	INSTANCE_OF LENGTH	BINDR - Binary Number Field 16

	MINVAL	6
	MAXVAL	32,767
	NOTE	Specifies the length of a request structure, including the length field. If it is necessary to include more data than the maximum allows, or if it is desirable to break a large structure into smaller pieces, set the high-order bit of the length field to 1. This indicates that the structure is continued in the next structure transmitted.
	REQUIRED	
ddmid	INSTANCE_OF REQUIRED	DDMID - DDM Identifier
format	INSTANCE_OF	DSSFMT - Data Stream Structure Format
	ENUVAL	X'01'
	NOTE	A request message.
	ENUVAL	X'41'
	NOTE	A request message, chained; do not continue on error. Next DSS has a different request correlator (RQSCRR).
	ENUVAL	X'51'
	NOTE	A request message, chained; do not continue on error. Next DSS has the same request correlator (RQSCRR).
	ENUVAL	X'61'
	NOTE	A request message, chained: continue on error. Next DSS has a different request correlator (RQSCRR).
	ENUVAL	X'71'
	NOTE	A request message, chained: continue on error. Next DSS has the same request correlator (RQSCRR).
	ENUVAL	X'05'
	NOTE	A request message with no expected reply.
	MINLVL	5
	ENUVAL	X'45'
	NOTE	A request message with no expected reply, chained; do not continue on error. Next DSS has a different request correlator (RQSCRR).
	MINLVL	5
	ENUVAL	X'55'
	NOTE	A request message with no expected reply, chained; do not continue on error. Next DSS has the same request correlator (RQSCRR).
	MINLVL	5
	ENUVAL	X'65'
	NOTE	A request message with no expected reply, chained; continue on error. Next DSS has a different request correlator (RQSCRR).

	MINLVL	5
	ENUVAL	X'75'
	NOTE	A request message with no expected reply, chained; continue on error. Next DSS has the same request correlator (RQSCRR).
	MINLVL REQUIRED	5
rqscrr	INSTANCE_OF NOTE	RQSCRR - Request Correlation Identifier Specifies a request identifier assigned by the source communications manager. This identifier is the same for all DSSs containing a request or containing data associated with a request the source communications manager sends. The same identifier is also specified for all DSSs containing replies to or data associated with that request.
	REQUIRED	
data	MINLEN	0
	MAXLEN	32761
	NOTE	An instance of a command is mapped onto this data area. The codepoint of the command identifies the class of the command.
	REQUIRED	
clscmd	NIL	
inscmd	NIL	

## SEE ALSO

<b>insvar</b>	<i>OBJDSS</i> (on page 536) <i>PRCCNVCD</i> (on page 621) <i>SYNERRCD</i> (on page 985)
<b>Semantic</b>	<i>ACCRDB</i> (on page 42) <i>ACCSEC</i> (on page 52) <i>AGNCMDPR</i> (on page 64) <i>APPCMNFL</i> (on page 68) <i>APPCMNI</i> (on page 72) <i>APPCMNT</i> (on page 76) <i>APPSRCCD</i> (on page 79) <i>APPSRCCR</i> (on page 86) <i>APPSRCER</i> (on page 91) <i>APPTRGER</i> (on page 95) <i>BGNATMCHN</i> (on page 107) <i>BGNBND</i> (on page 110) <i>CLASS</i> (on page 158) <i>CLSQR</i> (on page 165) <i>CMNAPPC</i> (on page 184) <i>CMNMGR</i> (on page 196) <i>CMNSYNCPT</i> (on page 202) <i>CMNTCPIP</i> (on page 214) <i>COMMAND</i> (on page 240) <i>DRPPKG</i> (on page 293)

*DSCRDBTBL* (on page 300)  
*DSS* (on page 308)  
*ENDATMCHN* (on page 333)  
*ENDBND* (on page 336)  
*EXCSAT* (on page 363)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSET* (on page 377)  
*EXCSQLSTT* (on page 381)  
*INHERITANCE* (on page 437)  
*INTRDBRQS* (on page 445)  
*OPNQRY* (on page 555)  
*PRPSQLSTT* (on page 636)  
*RDBCMM* (on page 728)  
*RDBOVR* (on page 740)  
*RDBRLLBCK* (on page 745)  
*REBIND* (on page 753)  
*RQSCRR* (on page 772)  
*SECCHK* (on page 800)  
*SYNCCTL* (on page 915)  
*SYNCMNBK* (on page 924)  
*SYNCMNCM* (on page 926)  
*SYNCMNFL* (on page 928)  
*SYNCMNI* (on page 931)  
*SYNCMNT* (on page 935)  
*SYNCRSY* (on page 982)  
*TCPCMNFL* (on page 992)  
*TCPCMNI* (on page 994)  
*TCPSRCCD* (on page 1007)  
*TCPSRCCR* (on page 1010)  
*TCPSRCER* (on page 1013)  
*TCPTRGER* (on page 1015)

**NAME**

RSCLMTRM — Resource Limits Reached

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1233'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

Resource Limits Reached (RSCLMTRM) Reply Message indicates that the requested command could not be completed due to insufficient target server resources. Examples of resource limitations are:

- The target agent has insufficient memory to keep track of DCLFIL collections.
- The lock manager cannot obtain another lock.
- The communications manager send or receive buffer overflowed.
- The target server lacks the memory or storage resource to create the instance of the manager requested. For example, an ACCRDB command could not create a target SQLAM manager because of the target server resource limitations.

Relational database (RDB) resource limitations are not reported with this reply message. The RDB resource limitations are reported through SQLCARD objects.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'1233'	
<b>csrposst</b>	INSTANCE_OF OPTIONAL NOTE  MINLVL	CSRPOSST - Cursor Position Status  This parameter is not returned by commands that operate on RDBs.  3
<b>dclnam</b>	INSTANCE_OF OPTIONAL NOTE NOTE  MINLVL	DCLNAM - Declared Name  Required if the file was accessed with DCLNAM. This parameter is not returned by commands that operate on RDBs.  3
<b>drcnam</b>	INSTANCE_OF MINLVL OPTIONAL NOTE	DRCNAM - Directory Name 2  Required when DRCNAM was specified on the command.



	NOTE	This parameter is not returned by commands that operate on RDBs.
	MINLVL	3
dtalckst	INSTANCE_OF OPTIONAL	DTALCKST - Data Lock Status
	NOTE	This parameter is not returned by commands that operate on RDBs.
	MINLVL	3
filnam	INSTANCE_OF OPTIONAL	FILNAM - File Name
	NOTE	Required when FILNAM was specified on the command.
	NOTE	This parameter is not returned by commands that operate on RDBs.
	MINLVL	3
prdid	INSTANCE_OF MINLVL OPTIONAL	PRDID - Product-specific Identifier
		3
rdbnam	INSTANCE_OF MINLVL OPTIONAL	RDBNAM - Relational Database Name
	NOTE	Required when RDBNAM was specified on the command.
	NOTE	This parameter is not returned by commands that operate on files.
reccnt	INSTANCE_OF MINVAL OPTIONAL	RECCNT - Record Count
	NOTE	Required for requests to insert multiple records in a file.
	NOTE	This parameter is not returned by commands that operate on RDBs.
	MINLVL	3
rscnam	INSTANCE_OF MINLVL OPTIONAL	RSCNAM - Resource Name Information
		3
rsctyp	INSTANCE_OF MINLVL OPTIONAL	RSCTYP - Resource Type Information
		3
rsncod	INSTANCE_OF MINLVL OPTIONAL	RSNCOD - Reason Code Information
		3
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
svrcod	INSTANCE_OF	SVRCOD - Severity Code

	REQUIRED
	ENUVAL 4 - WARNING - Warning Severity Code
	MINLVL 2
	NOTE Can be used if wild-card characters are specified.
	ENUVAL 8 - ERROR - Error Severity Code
	ENUVAL 16 - SEVERE - Severe Error Severity Code
	ENUVAL 32 - ACCDMG - Access Damage Severity Code
	ENUVAL 64 - PRMDMG - Permanent Damage Severity Code
	ENUVAL 128 - SESDMG - Session Damage Severity Code
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

## SEE ALSO

<b>cmdrpy</b>	<i>ACCRDB</i> (on page 42)
	<i>ACCSEC</i> (on page 52)
	<i>BGNATMCHN</i> (on page 107)
	<i>BGNBND</i> (on page 110)
	<i>BNDSQLSTT</i> (on page 136)
	<i>CLSQR</i> (on page 165)
	<i>CNTQR</i> (on page 222)
	<i>DRPPKG</i> (on page 293)
	<i>DSCRDBTBL</i> (on page 300)
	<i>DSCSQLSTT</i> (on page 304)
	<i>ENDATMCHN</i> (on page 333)
	<i>ENDBND</i> (on page 336)
	<i>EXCSAT</i> (on page 363)
	<i>EXCSQLIMM</i> (on page 371)
	<i>EXCSQLSET</i> (on page 377)
	<i>EXCSQLSTT</i> (on page 381)
	<i>INTRDBRQS</i> (on page 445)
	<i>OPNQR</i> (on page 555)
	<i>PRPSQLSTT</i> (on page 636)
	<i>RDBCMM</i> (on page 728)
	<i>RDBRLLBCK</i> (on page 745)
	<i>REBIND</i> (on page 753)
	<i>SECCHK</i> (on page 800)
<b>rpydta</b>	<i>COMMAND</i> (on page 240)
<b>Semantic</b>	<i>DSS</i> (on page 308)
	<i>RSCTYP</i> (on page 782)
	<i>SYNCPTOV</i> (on page 944)

**NAME**

RSCNAM — Resource Name Information

**DESCRIPTION (Semantic)****Dictionary** QDDBASD**Codepoint** X'112D'**Length** \***Class** CLASS**Sprcls** STRING - String

Resource Name Information (RSCNAM) String specifies the product specific name of a resource.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
<b>length</b>	*
<b>class</b>	X'112D'
<b>value</b>	INSTANCE_OF CHRSTRDR - Character String
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO****insvar** *RSCLMTRM* (on page 778)

**NAME**

RSCTYP — Resource Type Information

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'111F'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Resource Type Information (RSCTYP) String specifies the type of resource that reached its limit and sends an RSCLMTRM in response to a command.

The value STGLMT means that the target system has reached the limit of its memory resources.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	6	
<b>class</b>	X'111F'	
<b>value</b>	INSTANCE_OF	CODPNTDR - Codepoint Data Representation
	ENUVAL	X'1409' - STGLMT - Storage Limit Reached
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *RSCLMTRM* (on page 778)

## NAME

RSLSETFLG — Result Set Flags

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'2142'  
**Length** \*  
**Class** CLASS  
**Sprcls** BITSTRDR - Bit String Field

Result Set Flags (RSLSETFLG) specifies whether the requester desires the target SQLAM to return descriptive information about each column of the result set and the type of descriptive information to return. A standard, extended, or light descriptor can be requested. Result sets are generated by the EXCSQLSTT command that invokes a stored procedure. Each bit is a boolean flag.

- 0-2 *unused* Flags are no longer used and their value should be zero (B'000').
- 3 *dsconly* The requester requires the target SQLAM to return an FD:OCA description of the answer set data but does not expect the command to return any answer set data in the response to the EXCSQLSTT. The return of answer set data in the response to the EXCSQLSTT may, in fact, result in the incorrect operation of the application invoking the stored procedure.
- 4 *extended* Identifies the type of FD:OCA SQLDA descriptor returned to describe the answer set in order to ensure the correct operation of the application invoking the stored procedure.
- 00 The standard SQLDA is requested to describe each column returned in the answer set.
  - 01 The extended SQLDA is requested to describe each column returned in the answer set.
  - 10 The light SQLDA is requested to describe each column returned in the answer set.

The SQLDA types are defined by the SQLCINRD FD:OCA descriptor. Refer to the DRDA Reference for a description of an SQLCINRD.

**Intermediate System Processing**

The RSLSETFLG parameter, when defined for a given command, specifies both the type of the data to be described (input or output) and the amount of descriptor information to be returned (standard, extended, or light) with the command. All SQLAM levels that support the RSLSETFLG parameter support the return of descriptor information for output data.

If the SQLAM manager level of the downstream server does not support the RSLSETFLG option, then no RSLSETFLG parameter should be sent to the downstream server and no SQLCINRD will be returned by the intermediate server to the upstream requester.

If the SQLAM manager level of the downstream server does not support the return of descriptor information for the type of data requested by the upstream requester (input), then the RSLSETFLG option sent to the downstream server should request no descriptor information and no SQLCINRD will be returned by the intermediate server to the upstream requester.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'2142'	
dsconly	INSTANCE_OF	BITDR - A Single Bit
	LENGTH	1
	ENUVAL	B'0'
	NOTE	The requester is capable of processing answer set data returned in the response to EXCSQLSTT.
	ENUVAL	B'1'
	NOTE	The requester is not capable of processing answer set data returned in the response to EXCSQLSTT.
extended	INSTANCE_OF	BITDR - A Single Bit
	LENGTH	2
	ENUVAL	B'00'
	NOTE	Request a standard SQLDA to describe each column of the result set.
	ENUVAL	B'01'
	NOTE	Request an extended SQLDA to describe each column of the result set.
	ENUVAL	B'10'
	NOTE	Request a light SQLDA to describe each column of the result set.
	ENUVAL	B'11'
	NOTE	Not defined.
RESERVED		
MINLVL	7	
rest	INSTANCE_OF	BITDR - A Single Bit
	LENGTH	2
	RESERVED	
unused	INSTANCE_OF	BITDR - A Single Bit
	LENGTH	3
	RESERVED	
	MINLVL	7
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

<b>insvar</b>	<i>EXCSQLSTT</i> (on page 381)
<b>Semantic</b>	<i>EXCSQLSTT</i> (on page 381)

**NAME**

RSLSETRM — RDB Result Set Reply Message

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2219'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

RDB Result Set Reply Message (RSLSETRM) indicates that an EXCSQLSTT command invoked a stored procedure, that the execution of the stored procedure generated one or more result sets, and that additional information about these result sets follows the SQLCARD or SQLDTARD in the reply data of the response.

**DSS Carrier:** RPYDSS

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'2219'	
<b>pkgsnlst</b>	INSTANCE_OF REQUIRED	PKGSNLST - RDB Package Name, Consistency Token, and Section Number List
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF ENUVAL REQUIRED	SVRCOD - Severity Code 0 - INFO - Information Only Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**cmdrpy** *EXCSQLSTT* (on page 381)

**Semantic** *CNTQRY* (on page 222)  
*EXCSQLSTT* (on page 381)  
*LMTBLKPRC* (on page 475)  
*QRYBLK* (on page 670)

**NAME**

RSNCOD — Reason Code Information

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1127'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Reason Code Information (RSNCOD) String specifies a product-specific reason code.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	8	
class	X'1127'	
value	INSTANCE_OF	BYTSTRDR - Byte String
	LENGTH	4
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *RSCLMTRM* (on page 778)



**NAME**

RSYNCMGR — Resynchronization Manager

**DESCRIPTION (Semantic)****Dictionary** QDDBASD**Codepoint** X'14C1'**Length** \***Class** CLASS**Sprcls** MANAGER - Resource Manager

Sync Point Resynchronization Manager (RSYNCMGR) is a manager object of DDM that performs resynchronization for in-doubt units of work after a sync point operation failure. Also, in conjunction with the SYNCPTMGR provides resync server support by supporting the SYNCCTL migrate command.

A resync manager performs resynchronization using the SYNCRSY command. Resynchronization occurs by exchanging unit of work state information between resync managers. Resync manager includes, but is not limited to, the following:

1. Keeping track of unit of work states after a sync point failure
2. Initiating resynchronization protocols for any unit of work that may be in the in-doubt state because of a system or communications failure
3. Providing logging support for source SYNCPTMGRs without a log

For more information, see *SYNCPTOV* (on page 944).

The role of the RSYNCMGR at DDM Level 5 is illustrated in Figure 3-73 (on page 788). Resync operations flow as DDM commands, objects, and replies using either the TCP/IP or the LU 6.2 communication manager. The agent forwards resync commands and objects to the RSYNCMGR which then interfaces to the SYNCPTMGR or AGENT to perform resynchronizations. Resync replies are sent from the RSYNCMGR to the AGENT in the form of DDM reply and DDM objects which are sent by the AGENT using the communication manager to the remote AGENT.

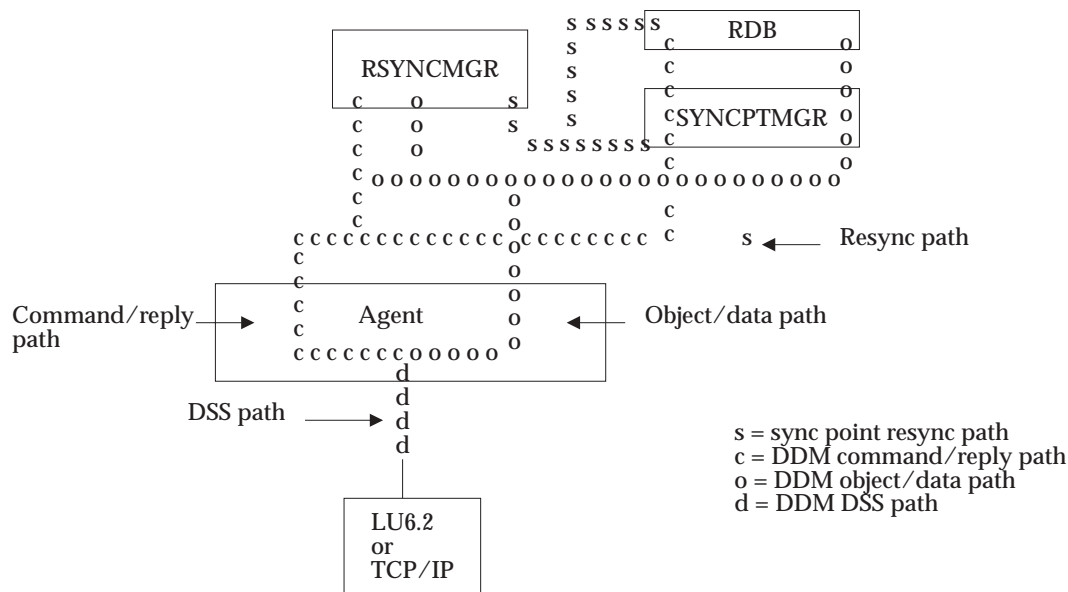


Figure 3-73 Server Paths for RSYNCMGR at DDM Level 5

The sync point manager (SYNCPTMGR) interfaces with the AGENT to perform resynchronization.

**Manager-Level Compatibility**

Table 3-10 illustrates the function of the SYNCPTMGR as it has grown and changed through the levels of DDM architecture.

Table 3-10 Sync Point Manager-Level Compatibility

DDM Levels	1	2	3	4	5
RSYNCMGR					5
<i>Manager Dependencies</i>					
SYNCPTMGR (resync server support only)					5
SYNCPTMGR (resynchronization only)					0
CMNAPPC					3
CMNTCPIP					5
AGENT					5

RSYNCMGR Level 5 uses DDM commands to perform resynchronization and can be used with SNA LU 6.2 or TCP/IP. See *SYNCP TOV* (on page 944).

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>NIL</b>
<b>clscmd</b>	<b>NIL</b>

<b>inscmd</b>	<b>NIL</b>	
<b>mgrlvl</b>	4	
<b>mgrdepls</b>	<b>MANAGER DEPENDENCY LIST</b>	
X'1444'	INSTANCE_OF	CMNAPPC - LU 6.2 Conversational Communications Manager
	MGRLVLN	3
X'1474'	INSTANCE_OF	CMNTCPIP - TCP/IP Communication Manager
	MGRLVLN	5
X'14C0'	INSTANCE_OF	SYNCPTMGR - Sync Point Manager
	MGRLVLN	5
X'1403'	INSTANCE_OF	AGENT - Agent
	MGRLVLN	5
<b>vldattls</b>	<b>VALID ATTRIBUTES</b>	
X'0019'	INSTANCE_OF	HELP - Help Text
X'1452'	INSTANCE_OF	MGRNAM - Manager Name
X'0045'	INSTANCE_OF	TITLE - Title

**SEE ALSO**

<b>insvar</b>	<i>DEPERRCD</i> (on page 270) <i>MGRLVL</i> (on page 506)
<b>mgrdepls</b>	<i>SYNCPTMGR</i> (on page 939)
<b>Semantic</b>	<i>AGENT</i> (on page 61) <i>SQLAM</i> (on page 847) <i>SYNCPTMGR</i> (on page 939) <i>SYNCPTOV</i> (on page 944)

**NAME**

RSYNCTYP — Resync Type

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'11EA'

**Length** \*

**Class** CLASS

**Sprcls** NAME - Name

Resync Type (RSYNCTYP) specifies the type of resync request or reply data.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	5	
class	X'11EA'	
value	INSTANCE_OF	BYTDR - An 8-bit Data Representation Value
	ENUVAL	X'01' - UOW indicates a resync request or resync reply that exchanges unit of work information.
	ENUVAL	X'02' - FORGET is an acknowledgement that all resync requests and replies have been received.
	ENUVAL	X'03' - END terminates a series of resync commands.
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *SYNCRRD* (on page 981)  
*SYNCRSY* (on page 982)

**NAME**

RTNEXTALL — Return all EXTDTAs for QRYDTAs Sent

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Length** \*

**Class** CONSTANT

On a CNTQRY command, RTNEXTALL indicates that the source system expects to receive all the EXTDTA objects associated with QRYDTA objects returned in response to the CNTQRY command.

---

value 2

**SEE ALSO**

**insvar** *RTNEXTDTA* (on page 792)

**Semantic** *QRYBLK* (on page 670)  
*QRYROWSET* (on page 697)

**NAME**

RTNEXTDTA — Return of EXTDTA Option

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2148'**Length** \***Class** CLASS**Sprcls** STRING - String

For a query using the limited block protocol, the RTNEXTDTA is specified on a CNTQRY command to indicate how EXTDTAs associated with the base data being retrieved by this command are to be returned. The option has effect until all EXTDTAs associated with the base data are retrieved by the application requester.

This option is ignored if there is no EXTDTA object associated with the reply to the CNTQRY command.

This option is ignored if the CNTQRY command does not retrieve any base data.

This option is ignored if the query is not using the limited block protocol (LMTBLKPRC).

<b>clsvar</b>	NIL	
<b>insvar</b>	INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'2148'	
<b>value</b>	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	8
	ENUVAL11 RTNEXTROW	
	NOTE	Return EXTDTAs by Row
	ENUVAL	2 RTNEXTALL
	NOTE	Return all EXTDTAs for QRYDTAs Sent
	DFTVAL	1 RTNEXTROW
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO****insvar** CNTQRY (on page 222)

**Semantic** CNTQRY (on page 222)  
 QRYBLK (on page 670)  
 QRYROWSET (on page 697)

**NAME**

RTNEXTROW — Return EXTDTAs by Row

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Length** \*

**Class** CONSTANT

On a CNTQRY command, RTNEXTROW indicates that the source system expects to receive externalized data objects from the target separately from the base data and one row at a time if the query can return any such objects. Each CNTQRY command can return data in one of the following ways:

- One or more QRYDTA objects if there are no EXTDTAs associated with any rows in the QRYDTA objects. The next CNTQRY command retrieves additional base data.
- One or more QRYDTA objects if there are EXTDTAs associated with one or more rows in the QRYDTA objects. Each subsequent CNTQRY command retrieves the EXTDTAs for the next row having associated EXTDTAs until all pending EXTDTAs are retrieved by the application requester.
- One or more EXTDTA objects for the next row in a previously returned QRYDTA objects.

See *CNTQRY* (on page 222) for details.

---

value	1
-------	---

**SEE ALSO**

**insvar** *RTNEXTDTA* (on page 792)

## NAME

RTNSETSTT — Return SET Statement

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD**Codepoint** X'210E'**Length** \***Class** CLASS**Sprcls** STRING - String

Return SET statement (RTNSETSTT) controls whether the target server must return one or more SQLSTT reply data objects, each containing an SQL SET statement for a special register whose setting has been modified on the current connection, upon successful processing of the command, if any special register has had its setting modified during execution of the command. No SQLSTT reply data object is returned by the target server if no special register has had its setting modified on the current connection, regardless of the RTNSETSTT setting.

In the event that a subsequent communications failure occurs, and the source server manages to reestablish a connection to the RDB at either the original or an alternate failover target server, such statements can be sent to the server via an EXCSQLSET command in order to restore the execution environment for the new connection.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	5	
<b>class</b>	X'210E'	
<b>value</b>	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	MINLVL	7
	LENGTH	2
	ENUVAL	X'00'
	NOTE	Target server must not return any SQL SET statements.
	ENUVAL	X'01'
	NOTE	Target server must return one or more SQL SET statements for special registers whose settings have been modified on the current connection if any special register has had its setting modified during execution of the command.
	DFTVAL	X'00'
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

## SEE ALSO

**insvar** *BGNATMCHN* (on page 107)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSET* (on page 377)  
*EXCSQLSTT* (on page 381)  
*RTNSETSTT*



**NAME**

RTNSQLDA — Return SQL Descriptor Area

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2116'

**Length** \*

**Class** CLASS

**Sprcls** BOOLEAN - Logical Value

Return SQL Descriptor Area (RTNSQLDA) Boolean State controls whether to return an SQL descriptor area that applies to the SQL statement the command identifies. The target SQLAM obtains the SQL descriptor area performing an SQL DESCRIBE function on the statement after the statement has been prepared.

The value TRUE indicates that the SQLDA is returned.

The value FALSE indicates that the SQLDA is not returned.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	5	
class	X'2116'	
value	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	LENGTH	2
	ENUVAL	X'F1' - TRUE - True State
	NOTE	Return the SQLDA.
	ENUVAL	X'F0' - FALSE - False State
	NOTE	Do not return the SQLDA.
	DFTVAL	X'F0' - FALSE - False State
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *EXCSQLSTT* (on page 381)  
*OPNQRY* (on page 555)  
*PRPSQLSTT* (on page 636)

**rpydta** *PRPSQLSTT* (on page 636)

**Semantic** *EXCSQLSTT* (on page 381)  
*OPNQRY* (on page 555)  
*PRPSQLSTT* (on page 636)  
*TYPSQLDA* (on page 1034)

NAME

SCALAR — Scalar Object

DESCRIPTION (Semantic)

**Dictionary** QDDPRMD

**Codepoint** X'0034'

**Length** \*

**Class** CLASS

**Sprcls** OBJECT - Self-identifying Data

Scalar Object (SCALAR) describes the abstract class of all scalar objects.

A scalar is a byte string onto which one or more data values have been mapped. Examples of simple scalars (having only a single data value) are binary numbers, character strings, boolean, and names.

Instances of mapped scalars must be encoded according to the sequence of the instance variables for the class. For example, the DATDR must be encoded *year, month, day, hour, minute, second, millisecond, microsecond, nanosecond* for a date and time or *year, month, day* for just a date. Another example is the MGRVLV which must be encoded *cdpnt, mgrlvln*.

All scalars are mapped onto the DDM data stream as shown in Figure 3-74.



Figure 3-74 Scalar to Data Stream Mapping

- Length is the total length of the scalar, including the length, codepoint, and data fields. See *DSS* (on page 308) for a discussion of:
  - Object segmentation in the data stream
  - Objects with the data greater than 5 bytes less than 32K bytes (32763 bytes) in length
- Codepoint is a unique identifier assigned to all instances of the same class of scalar.
- Scalar data consists of data in whatever form the class description requires of the scalar.

**Literal Form**

The literal form of all scalars is:

```
scalar_class_name(scalar_value)
```

where the *scalar\_value* is specified as a literal of the class which is required for the value of the named scalar class.

For example, the literal form of the count of the records in a file is:

```
RECCNT(153)
```

and the literal form of the title of a file in a file is:

```
TITLE('January employee payroll file')
```

**Note:** When reading class descriptions of scalar terms, the length and codepoint are always the first two data fields. If the length specified for the term is \*, then the actual length of the entire scalar object must be substituted for the \*.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
<b>length</b>	<b>4</b>
<b>class</b>	<b>X'0034'</b>
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

## SEE ALSO

**Semantic**

- ATMIND* (on page 104)
- BOOLEAN* (on page 142)
- CMDSRCID* (on page 178)
- CMMTYP* (on page 183)
- CNSVAL* (on page 221)
- COLLECTION* (on page 238)
- DFTVAL* (on page 276)
- DSS* (on page 308)
- ENDCHNTYP* (on page 341)
- ENUVAL* (on page 347)
- EXTDTA* (on page 397)
- FDOOTA* (on page 408)
- FDOEXT* (on page 410)
- FDOOFF* (on page 412)
- INHERITANCE* (on page 437)
- IPADDR* (on page 450)
- MAXVAL* (on page 499)
- MGRVLVLS* (on page 508)
- MGRVLVLOVR* (on page 511)
- MINVAL* (on page 519)
- OBJECT* (on page 540)
- OBJOVR* (on page 544)
- OOPOVR* (on page 547)
- PKGNAME* (on page 594)
- PKGNAMECSN* (on page 596)
- PKGNAMECT* (on page 598)
- PKTOBJ* (on page 608)
- PLGINCNT* (on page 610)
- QRYINSID* (on page 688)
- RESERVED* (on page 762)
- SNAADDR* (on page 827)
- SPCVL* (on page 831)
- SRVLCNT* (on page 875)
- SRVPTY* (on page 883)
- SYNCTYPE* (on page 984)
- TCPHOST* (on page 999)
- TCPPTHOST* (on page 1006)
- UOWDSP* (on page 1037)
- UOWID* (on page 1038)

*UOWSTATE* (on page 1040)  
*XIDCNT* (on page 1111)

**NAME**

SCLDTALEN — Scalar Data Length

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'0100'

**Length** \*

**Class** CLASS

**Sprcls** FIELD - A discrete unit of data

The Scalar Data Length (SCLDTALEN) Field specifies the length of the instance variable that immediately follows. The instance variables for which SCLDTALEN can be used are as follows:

- RDB Collection Identifier (RDBCOLID)
- Relational Database Name (RDBNAM)
- RDB Package Identifier (PKGID)

Also, the SCLDTALEN can only be used when the aforementioned instance variables are inside the following scalar objects:

- RDB Package Name (PKGNAM)
- RDB Package Name, Consistency Token, and Section Number (PKGNAMCSN)
- RDB Package Name and Consistency Token (PKGNAMCT)

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
scldtalen	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	16
	MINVAL	1
	MINLVL	7
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

None.

**NAME**

SECCHK — Security Check

**DESCRIPTION (Semantic)****Dictionary** QDDBASD**Codepoint** X'106E'**Length** \***Class** CLASS**Sprcls** COMMAND - Command

The Security Check (SECCHK) Command sends information to the target security manager to authenticate the user.

**Source System Processing**

The source system determines the location of the security manager:

Local Call the local security server.

Remote Send the SECCHK command to the remote security server.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**DSS Carrier: RQSDSS****Target System Processing**

The ACCSEC command must always precede the SECCHK command when any of the valid security mechanisms are active. Table 3-11 shows how the selected security mechanisms control the parameters or data objects sent on the SECCHK command. A number indicates the number of occurrences that are required.

**Table 3-11** SECCHK Valid Security Mechanism Combinations

	USRID	PASSWORD	NEWPASSWORD	SECTKN
DCESEC				X
USRIDPWD	X	X		
USRIDONL	X			
USRIDNWPWD	X	X	X	
USRSBSPWD	X			1
UERENCPWD	X			1
EUSRIDPWD				2
EUSRIDNWPWD				3
KERSEC				1
EUSRIDDTA				1
EUSRPWDDTA				2
EUSRNPWDDTA				3
PLGIN				1

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**Exceptions**

If the security information is acceptable, then the SECCHKRM is returned specifying the SECCHKCD of 00 and the SVRCOD is set to INFO.

If the security information is not acceptable at the target server, then the SECCHKRM with the appropriate SECCHKCD value is returned, and the SVRCOD is set higher than INFO.

**Source System Reply Processing**

Return any reply messages to the requester.

If the target SECTKN is not acceptable to the source server, then the source server must terminate the network connection and return a security error to the user.

**Handling Subsequent SECCHK Commands**

The source server can send repeated SECCHK commands when it desires to reuse a connection for another application or transaction. The SECCHK must be preceded by an EXCSAT and ACCSEC, and followed by an ACCRDB.

**Data Conversion**

If the CCSIDMGR is active and the code pages between the source and target systems are different, then the character data within the DDM parameter objects undergoes conversion between the code pages.

**SECMEC and SECTKN Rules**

If the SECMEC value is DCESEC or PLGIN, then the *sectkn* is required and must be sent. Otherwise, the *sectkn* should not be sent. A *sectkn* may also be returned as reply data to a SECCHK command if the SECMEC value is DCESEC and DCE is using mutual authentication or the SECMEC value is PLGIN. (See *DCESECOVR* (on page 253) for more details on DCESEC, and see *PLGINOVR* (on page 615) for more details on PLGIN.)

clsvar	NIL	
insvar	CLASS INSTANCE VARIABLES	
length	*	
class	X'106E'	
newpassword	INSTANCE_OF OPTIONAL NOTE	NEWPASSWORD - New Password  If the security mechanism is USRIDNWPWD, then NEWPASSWORD is REQUIRED.
password	INSTANCE_OF OPTIONAL NOTE	PASSWORD - Password  If the security mechanism is USRIDPWD or USRIDNWPWD, then PASSWORD is REQUIRED. PASSWORD may contain an RACF passticket.
plginid	INSTANCE_OF OPTIONAL NOTE  MINLVL	PLGINID - Security Plug-in-specific Identifier  This parameter is used to indicate the optionally provided plug-in module-specific identifier. 7
plginnm	INSTANCE_OF OPTIONAL REQUIRED NOTE  MINLVL	PLGINNM - Security Plug-in Name  If SECMEC is PLGIN. This parameter is used to indicate the plug-in module to be used if the security mechanism is PLGIN. 7
rdbnam	INSTANCE_OF OPTIONAL NOTE	RDBNAM - Relational Database Name  This parameter may be required if the target server owns multiple RDBs, and allows each RDB to have its own security mechanism.
secmec	INSTANCE_OF REQUIRED NOTE  NOTE	SECMEC - Security Mechanism  The SECMEC parameter is always sent to identify the contents of the SECTKN parameter, and if the USRID parameter is needed.  The SECMEC parameter is always sent to identify the contents of SECTKN, and to indicate which of the USRID, PASSWORD, and NEWPASSWORD parameters are needed.



secmgrnm	INSTANCE_OF OPTIONAL IGNORABLE NOTE  CMDTRG	SECMGRNM - Security Manager Name  This parameter has a null value and does not have to be validated.
sectkn	INSTANCE_OF REPEATABLE OPTIONAL NOTE  NOTE  NOTE  NOTE  NOTE  NOTE  NOTE  NOTE  NOTE  NOTE	SECTKN - Security Token  If the security mechanism is DCESEC, then the SECTKN is required. A <i>sectkn</i> may also be returned as reply data to a SECCHK command if the SECMEC value is DCESEC and DCE is using mutual authentication. See <i>DCESECOVR</i> (on page 253) for more details.  This parameter is required if the security mechanism is USRSBSPWD. See <i>USRSECOVR</i> (on page 1050) for details.  This parameter is required if the security mechanism is USRENCPWD. See <i>USRSECOVR</i> (on page 1050) for details.  Two SECTKNs are required for security mechanism EUSRIDPWD. See <i>USRSECOVR</i> (on page 1050) for details.  Three SECTKNs are required for security mechanism EUSRIDNWPWD. See <i>USRSECOVR</i> (on page 1050) for details.  If the security mechanism is USRIDONL, then the SECTKN is ignored.  This parameter is required if the security mechanism is KERSEC. See <i>KERSECOVR</i> (on page 466) for details.  One SECTKN is required for security mechanism EUSRIDDTA. See <i>EDTASECOVR</i> (on page 323) for details.  Two SECTKNs are required for security mechanism EUSRPWDDTA. See <i>EDTASECOVR</i> (on page 323) for details.  Three SECTKNs are required for security mechanism EUSRNPWDDTA. See <i>EDTASECOVR</i> (on page 323) for details.  This parameter is required if the security mechanism is PLGIN. See <i>PLGINOVR</i> (on page 615) for details.
usrid	INSTANCE_OF OPTIONAL	USRID - User ID at the Target System

	NOTE	USRID is required when the security mechanism is USRIDPWD, USRIDNWPWD, or USRIDONL.
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>cmddta</b>	<b>NIL</b>	
X'11DC'	INSTANCE_OF OPTIONAL NOTE	SECTKN - Security Token  If the security mechanism is DCESEC then the SECTKN is REQUIRED.
<b>rpydta</b>	<b>NIL</b>	
X'11DC'	INSTANCE_OF OPTIONAL NOTE	SECTKN - Security Token  Used if the security mechanism returns a security token that must be sent back to the source system.
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'121C'	INSTANCE_OF	CMDATHRM - Not Authorized to Command
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1250'	INSTANCE_OF	CMDNSPRM - Command Not Supported
X'123C'	INSTANCE_OF	INVRQSRM - Invalid Request
X'1218'	INSTANCE_OF	MGRDEPRM - Manager Dependency Error
X'1253'	INSTANCE_OF	OBJNSPRM - Object Not Supported
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'221A'	INSTANCE_OF NOTE	RDBAFLRM - RDB Access Failed Reply Message Only provided if RDBNAM is provided. When RDBAFLRM is returned, the target SQLAM instance is destroyed.
X'1233'	INSTANCE_OF	RSCLMTRM - Resource Limits Reached
X'1219'	INSTANCE_OF	SECCHKRM - Security Check
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'125F'	INSTANCE_OF	TRGNSPRM - Target Not Supported
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

**SEE ALSO**

<b>inscmd</b>	<i>SECMGR</i> (on page 814)
<b>insvar</b>	<i>PRCCNVCD</i> (on page 621)
<b>Semantic</b>	<i>ACCSEC</i> (on page 52) <i>ACCSECRD</i> (on page 58) <i>DCESECOVR</i> (on page 253) <i>DCESECTKN</i> (on page 257) <i>DHENC</i> (on page 280) <i>EUSRIDDTA</i> (on page 355) <i>EUSRIDNWPWD</i> (on page 357) <i>EUSRIDPWD</i> (on page 358)

*EUSRNPWDDTA* (on page 359)  
*EUSRPWDDTA* (on page 361)  
*PLGIN* (on page 609)  
*PLGINOVR* (on page 615)  
*PLGINSECTKN* (on page 620)  
*PWDENC* (on page 644)  
*PWDSBS* (on page 646)  
*SECCHKRM* (on page 809)  
*SECMGR* (on page 814)  
*SECTKN* (on page 820)  
*SYNCPTOV* (on page 944)  
*TCPCMNI* (on page 994)  
*USRSECOVR* (on page 1050)

NAME

SECCHKCD — Security Check Code

DESCRIPTION (Semantic)

**Dictionary** QDDBASD

**Codepoint** X'11A4'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

The Security Check Code (SECCHKCD) String codifies the security information and condition for the SECCHKRM.

Table 3-12 shows the relationship between the SECCHKCD parameter and the SVRCOD parameter in the SECCHKRM.

**Table 3-12** Relationship of SECCHKCD and SVRCOD in the SECCHKRM

SECCHKCD	SVRCOD
X'00'	INFO
X'01'	ERROR
X'02'	INFO
X'03'	ERROR
X'04'	ERROR
X'05'	INFO
X'06'	ERROR
X'07'	ERROR
X'08'	INFO
X'09'	ERROR
X'0A'	ERROR
X'0B'	ERROR
X'0E'	ERROR
X'0F'	ERROR
X'10'	ERROR
X'12'	ERROR
X'13'	ERROR
X'14'	ERROR
X'15'	ERROR

---

<b>clsvar</b>	NIL
<b>insvar</b>	CLASS INSTANCE VARIABLES
<b>length</b>	5
<b>class</b>	X'11A4'

---

value	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	LENGTH	2
	ENUVAL	X'00'
	NOTE	The security information is correct and acceptable.
	ENUVAL	X'01'
	NOTE	SECMEC value not supported.
	ENUVAL	X'02'
	NOTE	DCE informational status issued.
	ENUVAL	X'03'
	NOTE	DCE retryable error.
	ENUVAL	X'04'
	NOTE	DCE non-retryable error.
	ENUVAL	X'05'
	NOTE	GSSAPI informational status issued.
	ENUVAL	X'06'
	NOTE	GSSAPI retryable error.
	ENUVAL	X'07'
	NOTE	GSSAPI non-retryable error.
	ENUVAL	X'08'
	NOTE	Local Security Service informational status issued.
	ENUVAL	X'09'
	NOTE	Local Security Service retryable error.
	ENUVAL	X'0A'
	NOTE	Local Security Service non-retryable error.
	ENUVAL	X'0B'
	NOTE	SECTKN missing when it is required or it is invalid.
	ENUVAL	X'0E'
	NOTE	Password expired.
	ENUVAL	X'0F'
	NOTE	Password invalid.
	ENUVAL	X'10'
	NOTE	Password missing.
	ENUVAL	X'12'
	NOTE	User ID missing.
	ENUVAL	X'13'
	NOTE	User ID invalid.
	ENUVAL	X'14'
	NOTE	User ID revoked.
	ENUVAL	X'15'
	NOTE	New Password invalid.
	ENUVAL	X'16'
	NOTE	Authentication failed because of connectivity restrictions enforced by the security plug-in.
	ENUVAL	X'17'
	NOTE	Invalid GSS-API server credential.
	ENUVAL	X'18'

NOTE GSS-API server credential expired on the database server.

## REQUIRED

---

<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

---

## SEE ALSO

**insvar** *ACCSECRD* (on page 58)  
*SECCHKRM* (on page 809)

**Semantic** *ACCSECRD* (on page 58)  
*DCESECOVR* (on page 253)  
*EDTASECOVR* (on page 323)  
*PLGINOVR* (on page 615)  
*SECCHK* (on page 800)  
*SECCHKRM* (on page 809)  
*USRSECOVR* (on page 1050)

NAME

SECCHKRM — Security Check

DESCRIPTION (Semantic)

**Dictionary** QDDBASD

**Codepoint** X'1219'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

The Security Check (SECCHKRM) Reply Message indicates the acceptability of the security information. The security manager specifies the state of the security information via the SECCHKCD.

The relationship between the SECCHKCD and the SVRCOD is shown in SECCHKCD.

When mutual authentication of the source and target servers is requested, the *sectkn* must be returned. SECTKN flows as reply data to the SECCHK command. It must flow after the SECCHKRM message.

If the target SECTKN is not acceptable to the source server, then the source server must terminate the network connection and return a security error to the user.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'1219'	
<b>secchkcd</b>	INSTANCE_OF REQUIRED	SECCHKCD - Security Check Code
<b>sectkn</b>	INSTANCE_OF OPTIONAL IGNORABLE NOTE	SECTKN - Security Token  SECTKN is required when the selected security mechanism requires mutual authentication.
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svcerrno</b>	INSTANCE_OF NOTE  OPTIONAL	SVCERRNO - Security Service Error Number Error number from called service. SRVDGN may contain additional information.
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL ENUVAL	SVRCOD - Severity Code  0 - INFO - Information Only Severity Code 8 - ERROR - Error Severity Code

	ENUVAL	16 - SEVERE - Severe Error Severity Code
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

<b>cmdrpy</b>	<i>SECCHK</i> (on page 800)
<b>Semantic</b>	<i>DCESECOVR</i> (on page 253) <i>EDTASECOVR</i> (on page 323) <i>PLGIN</i> (on page 609) <i>PLGINOVR</i> (on page 615) <i>SECCHK</i> (on page 800) <i>SECCHKCD</i> (on page 806) <i>TCPCMNI</i> (on page 994) <i>USRSECOVR</i> (on page 1050)



**NAME**

SECMEC — Security Mechanism

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'11A2'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

DRDA defines a security mechanism, in the context of this term, as a combination of security mechanism components. To avoid confusion, however, the reader should be aware that throughout this specification the components themselves are often referred to as security mechanisms but that they are usually used in combination, and that the codes in the (SECMEC) string specify one or more combinations of security mechanisms components.

When SECMEC flows from the source server to the target server, the SECMEC parameter specifies the security mechanism combination that the source server wants to use.

When SECMEC flows from the target server to the source server, the SECMEC parameter must either reflect the value requested by the source server or if the target server does not support the requested security mechanism, then the target server returns the SECMEC values that it does support.

**Allowable Combinations of Security Mechanism Components**

Table 3-13 shows the combinations of the security mechanism components.

**Table 3-13** Valid Security Mechanism Component Combinations

<b>Combination ID</b>	<b>Value</b>	<b>Security Mechanism Components</b>
DCESEC	1	OSFDCE
USRIDPWD	3	USRIDSEC, PWDSEC
USRIDONL	4	USRIDSEC
USRIDNWPWD	5	USRIDSEC, PWDSEC, NWPWDSEC
USRSBSPWD	6	USRIDSEC, PWDSEC, PWDSBS
USRENCPWD	7	USRIDSEC, PWDSEC, PWDENC
USRSSBPWD	8	USRIDSEC, PWDSEC, PWDSSB
EUSRIDPWD	9	USRSECOVR
EUSRIDNWPWD	10	USRSECOVR
KERSEC	11	KERSECOVR
EUSRIDDTA	12	EDTASECOVR
EUSRPWDDTA	13	EDTASECOVR
EUSRNPWDDTA	14	EDTASECOVR
PLGIN	15	PLGINOVR

**Combination Rules Summary**

The PLGIN security mechanism component must be used by itself.

The KERSEC security mechanism component must be used by itself.

The OSFDCE security mechanism component must be used by itself.

The USRIDSEC security mechanism component can be used in combination with any of the PWDSEC, PWDENC, PWDSBS, or PWDSSB security mechanism components, with the PWDSEC and NWPWDSEC security mechanism components, or just by itself.

The PWDSEC, PWDSBS, PWDSSB, and PWDENC security mechanism components cannot be used alone. They require the USRIDSEC security mechanism component.

The NWPWDSEC security mechanism component cannot be used alone. It requires the USRIDSEC and PWDSEC security mechanism components.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'11A2'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	16
	ENUVAL	1 - DCESEC - Distributed Computing Environment
	ENUVAL	3 - USRIDPWD - User ID and Password Security Mechanism
	ENUVAL	4 - USRIDONL - User ID Only Security Mechanism
	ENUVAL	5 - USRIDNWPWD - User ID, Password, and New Password Security Mechanism
	ENUVAL	6 - USRSBSPWD - User ID with Substitute Password
	ENUVAL	7 - USRENCPWD - User ID with Encrypted Password
	ENUVAL	8 - USRSSBPWD - User ID with Strong Password Substitute
	ENUVAL	9 - EUSRIDPWD - Encrypted User ID and Password Security Mechanism
	ENUVAL	10 - EUSRIDNWPWD - Encrypted User ID, Password, New Password Security Mechanism
	ENUVAL	11 - KERSEC - Kerberos Security
	ENUVAL	12 - EUSRIDDTA - Encrypted User ID and Security-Sensitive Data Security Mechanism
	ENUVAL	13 - EUSRPWDDTA - Encrypted User ID, Password, and Security-sensitive Data Security Mechanism
	ENUVAL	14 - EUSRNPWDDTA - Encrypted User ID, Password, New Password, and Security-sensitive Data Security Mechanism
	ENUVAL	15 - PLGIN - Plug-in Security
	REPEATABLE	
	NOTE	The SECMEC may only contain multiple values when it is carried in the ACCSECRD and the target security manager does not support the requested security mechanism.
	REQUIRED	

---

<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

---

**SEE ALSO**

<b>insvar</b>	<i>ACCSEC</i> (on page 52) <i>ACCSECRD</i> (on page 58) <i>PRCCNVCD</i> (on page 621) <i>SECCHK</i> (on page 800) <i>SECCHKCD</i> (on page 806)
<b>Semantic</b>	<i>ACCSEC</i> (on page 52) <i>DCESECOVR</i> (on page 253) <i>EDTASECOVR</i> (on page 323) <i>PASSWORD</i> (on page 578) <i>PLGINOVR</i> (on page 615) <i>QRYDTA</i> (on page 686) <i>SECCHK</i> (on page 800) <i>SECOVR</i> (on page 819) <i>SQLATTR</i> (on page 854) <i>SQLCINRD</i> (on page 857) <i>SQLDARD</i> (on page 859) <i>SQLDTA</i> (on page 860) <i>SQLDTARD</i> (on page 862) <i>SQLRSLRD</i> (on page 867) <i>SQLSTT</i> (on page 868) <i>SQLSTTVRB</i> (on page 871) <i>USRSECOVR</i> (on page 1050)

## NAME

SECMGR — Security Manager

## DESCRIPTION (Semantic)

**Dictionary** QDDBASD**Codepoint** X'1440'**Length** \***Class** CLASS**Sprcls** MANAGER - Resource Manager

The Security Manager (SECMGR) is a basic operative part of the DDM architecture. Primarily, the security manager ensures that the requester, represented by the agent, is only allowed authorized access to files, commands, dictionaries, directories, and other objects. The security manager is also responsible for identification and authentication processing when these functions are not used or not provided by the communications facility.

The target agent, representing the requester, is assigned the security rights that the communications facilities established when communications initially began between the requester and target system. If the communication facilities do not provide any user security functions, the target agent is assigned *default* security rights. The default security rights may contain authorization to all *public* or *nonsecured* resources. The actual authorization the default security rights provide is implementation (or installation)-defined.

All of the reply messages end with *ATHRM*, which stands for *authorization reply message*.

**SNA LU 6.2 Security**

The SNA LU 6.2 security tower (set of functions) provides identification and authentication within the communications facility. Both node-to-node mutual authentication and requester authentication can be performed. For more information and details about SNA LU 6.2 security see the *SNA Transaction Programmer's Reference Manual for LU Type 6.2* (GC30-3084, IBM).

Three important elements of SNA LU 6.2 are the DES key for node-to-node authentication, the user identification, and the user password. When node-to-node authentication is used, the user password is not transmitted; instead a bit known as the *already verified indicator* is sent with the user identification.

**Other Forms of Network Security**

When the communications facilities do not provide identification and authentication, the DDM security manager must provide it. When the communications facilities do provide identification and authentication, the security manager may also provide this support. The DCE Security Mechanism (OSFDCE) is an example of a security mechanism used to provide identification and authentication. Other mechanisms which provide identification and authentication may be used.

**Kerberos Security Mechanism**

DDM provides support for utilizing the Kerberos security mechanism. See *KERSECOVR* (on page 466) for more information about Kerberos security.

**DCE Security Mechanism**

DDM provides support for utilizing the DCE-based security mechanism. See *DCESECOVR* (on page 253) for more information about OSF DCE security.

**PLGIN Security Mechanism**

DDM provides support for utilizing the plug-in-based security mechanism. See *PLGINOVR* (on page 615) for more information.

**User ID Security and Password Security with Optional Encryption**

See *USRSECOVR* (on page 1050) for the DDM commands and replies that flow in the normal process of establishing a connection while using user ID and password security mechanisms.

The following additional DDM Level 6 options for password security are provided: optional password encryption and password substitution. See *USRSECOVR* (on page 1050) for more details.

**User ID Security**

In some environments, the only security token required is the user ID to ensure proper authorization for accessing data in the target system. A source server acquires the user ID and sends it as part of the SECCHK command. The target server uses its security manager to identify and authenticate the end user to allow access to the target system's resources.

**Encrypted Data Security**

DDM provides support for encrypting user ID and security-sensitive data; user ID, password, and security-sensitive data; and user ID, password, new password, and security-sensitive data. See for more information.

**Security Manager-Level Protocol Errors**

The ACCSEC and SECCHK commands can be used only at specific times. The security commands are used immediately after the initializing EXCSAT command when the source server has requested the security manager at DDM manager Level 5 or higher; if the security commands are not used, then the target server responds with the PRCCNVRM for the command following the EXCSAT command. Note that when using TCP/IP, it can be assumed that the SECMGR is at least Level 5.

**Table 3-14** Security Manager-Level Compatibility

DDM Levels	1	2	3	4	5	6
SECMGR <sup>14</sup>	1	1	1	1	5	6
<i>Commands</i>						
ACCSEC					5	6
SECCHK					5	6

Note that the numbers 5 and 6 in the SECMGR row of the above table refer to the SECMGR level, whereas the 5 and 6 in the ACCSEC and SECCHK rows refer to the level of function of these commands at DDM Levels 5 and 6.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'1440'	
<b>clscmd</b>	NIL	
<b>inscmd</b>	INSTANCE COMMANDS	
X'106D'	INSTANCE_OF MINLVL REQUIRED	ACCSEC - Access Security 5
X'106E'	INSTANCE_OF MINLVL REQUIRED	SECCHK - Security Check 5
<b>mgrlvl</b>	6 NOTE MINLVL	The SECMGR can be at DDM Level 1 or DDM Level 5 or higher. 5
<b>mgrdepls</b>	NIL	
<b>vldattls</b>	VALID ATTRIBUTES	
X'0019'	INSTANCE_OF	HELP - Help Text
X'1196'	INSTANCE_OF	SECMGRNM - Security Manager Name
X'0045'	INSTANCE_OF	TITLE - Title

**SEE ALSO**

<b>insvar</b>	<i>DEPERRCD</i> (on page 270) <i>MGRLVL</i> (on page 506) <i>PRCCNVCD</i> (on page 621) <i>SUPERVISOR</i> (on page 908)
<b>mgrdepls</b>	<i>AGENT</i> (on page 61) <i>SQLAM</i> (on page 847)
<b>Semantic</b>	<i>ACCSEC</i> (on page 52) <i>AGENT</i> (on page 61) <i>DCESECOVR</i> (on page 253) <i>DCESECTKN</i> (on page 257) <i>EDTASECOVR</i> (on page 323) <i>EXTENSIONS</i> (on page 400) <i>INHERITANCE</i> (on page 437) <i>MANAGER</i> (on page 491) <i>MGRLVLN</i> (on page 509) <i>NEWPASSWORD</i> (on page 531)

---

14. The SECMGR can be at DDM Levels 1, 5, or 6.

*PASSWORD* (on page 578)  
*PLGINOVR* (on page 615)  
*PLGINSECTKN* (on page 620)  
*RDBOVR* (on page 740)  
*SECOVR* (on page 819)  
*SECTKNOVR* (on page 822)  
*SQLAM* (on page 847)  
*SUBSETS* (on page 902)  
*SYNCPTOV* (on page 944)  
*USRSECOVR* (on page 1050)

**NAME**

SECMGRNM — Security Manager Name

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD  
**Codepoint** X'1196'  
**Length** \*  
**Class** CLASS  
**Sprcls** MGRNAM - Manager Name

Security Manager Name (SECMGRNM) Manager Name specifies the name of a DDM server's security manager component. Only one instance of the security manager exists in a server, and it does not have an architected name.

SECMGRNM is specified as a null parameter (length equals 4 and no name value is specified).

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
length	4
class	X'1196'
name	LENGTH 0 REQUIRED
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO**

**insvar** ACCSEC (on page 52)  
 SECCHK (on page 800)  
**Semantic** DCESECOVR (on page 253)  
 USRSECOVR (on page 1050)  
**vldattls** SECMGR (on page 814)



**NAME**

SECOVR — Security Overview

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

The Security Overview (SECOVR) is a general discussion on the topic of security. Security is the means of protecting the customer's resources.

**DDM Architecture and Security**

Each DDM server (source and target) uses the security facilities of its system. The Security Manager (see *SECMGR* (on page 814)) Resource Manager is the logical image of the system security facilities. The DDM architecture specifies how to use security.

See also the following terms for more information on security:

DCESECOVR DCE Security Overview (see *DCESECOVR* (on page 253))

EDTASECOVR Encrypted Data Security Overview (see *EDTASECOVR* (on page 323))

PLGINOVR Plug-in Security (see *PLGINOVR* (on page 615))

SECMEC Security Mechanism (see *SECMEC* (on page 811))

SECMGR Security Manager (see *SECMGR* (on page 814))

SNASECOVR LU 6.2 Security Overview (see *SNASECOVR* (on page 828))

USRSECOVR User ID Security Overview (see *USRSECOVR* (on page 1050))

This overview discusses several security mechanism components associated with the USRID term, including those dealing with encryption.

**SEE ALSO**

**Semantic** *SECTKN* (on page 820)

NAME

SECTKN — Security Token

DESCRIPTION (Semantic)

**Dictionary** QDDBASD  
**Codepoint** X'11DC'  
**Length** \*  
**Class** CLASS  
**Sprcls** BYTSTRDR - Byte String

The Security Token (SECTKN) Byte String is information provided and used by the various security mechanisms. See *SECOVR* (on page 819) for information on the DDM flows that carry the SECTKN.

The security token contains security context information to identify and authenticate a user to the target server or to authenticate the target server to the source server.

Depending on the security mechanism, SECTKN may flow either as an object or as an instance variable with the ACCSEC and SECCHK commands. For example, SECTKN flows as an instance variable when the USRSBSPWD and USRENCPWD security mechanisms are chosen, whereas if the OSFDCE security mechanism is used, then SECTKN flows as an object because DCE security tokens can be arbitrarily large.

<b>clsvar</b>	NIL
<b>insvar</b>	CLASS INSTANCE VARIABLES
length	*
class	X'11DC'
value	INSTANCE_OF BYTSTRDR - Byte String REQUIRED
<b>clscmd</b>	NIL
<b>inscmd</b>	NIL

SEE ALSO

**cmdtda** *SECCHK* (on page 800)  
*EXCSQLSTT* (on page 381)  
*PRPSQLSTT* (on page 636)

**insvar** *ACCSEC* (on page 52)  
*ACCSECRD* (on page 58)  
*SECCHK* (on page 800)  
*SECCHKCD* (on page 806)  
*SECCHKRM* (on page 809)  
*SECTKNOVR* (on page 822)

**rpydta** *SECCHK* (on page 800)  
*EXCSQLSTT* (on page 381)  
*OPNQRY* (on page 555)

**Semantic** *ACCSEC* (on page 52)  
*DCESECOVR* (on page 253)  
*DHENC* (on page 280)  
*EDTASECOVR* (on page 323)

*EUSRIDDTA* (on page 355)  
*EUSRIDNWPWD* (on page 357)  
*EUSRIDPWD* (on page 358)  
*EUSRNPWDDTA* (on page 359)  
*EUSRPWDDTA* (on page 361)  
*PLGINOVR* (on page 615)  
*PLGINSECTKN* (on page 620)  
*PWDENC* (on page 644)  
*SECCHK* (on page 800)  
*SECCHKRM* (on page 809)  
*SECTKNOVR* (on page 822)  
*USRENCPWD* (on page 1043)  
*USRSBSPWD* (on page 1049)  
*USRSECOVR* (on page 1050)

NAME

SECTKNOVR — Security Token Override

DESCRIPTION (Semantic)

**Dictionary** QDDBASD  
**Codepoint** X'190B'  
**Length** \*  
**Class** CLASS  
**Sprcls** STRING - String

The Security Token Override (SECTKNOVR) sends two security tokens between SECMGRs. The Encryption seed is sent in the first SECTKN. The Encryption token is sent in the second SECTKN. The SECTKNOVR DSS carrier is encrypted. The encrypted SECTKNOVR is sent by the intermediate server with every request or reply containing encrypted objects for any of the following DDM commands: BNDSQLSTT, CNTQRY, DSCRDBTBL, DSCSQLSTT, EXCSQLIMM, EXCSQLSET, EXCSQLSTT, OPNQRY, PRPSQLSTT.

This is one type of behavior at an intermediate server to send SECTKNOVR. The intermediate server may choose to decrypt and rebuild the DSS according to the requirements of the connection.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'190B'	
<b>sectkn</b>	INSTANCE_OF REPEATABLE REQUIRED NOTE	SECTKN - Security Token  This parameter is required to be encrypted. Two SECTKNs are required. See EDTASECOVR for details.
	MINLVL	7
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

SEE ALSO

**cmdtda** *BNDSQLSTT* (on page 136)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSET* (on page 377)  
*EXCSQLSTT* (on page 381)  
*OPNQRY* (on page 555)  
*PRPSQLSTT* (on page 636)

**rpydta** *CNTQRY* (on page 222)  
*DSCRDBTBL* (on page 300)  
*DSCSQLSTT* (on page 304)  
*EXCSQLSTT* (on page 381)  
*OPNQRY* (on page 555)  
*PRPSQLSTT* (on page 636)

<b>Semantic</b>	<i>CNTQRY</i> (on page 222)
	<i>DSCRDBTBL</i> (on page 300)
	<i>DSCSQLSTT</i> (on page 304)
	<i>EDTASECOVR</i> (on page 323)
	<i>EUSRIDDTA</i> (on page 355)
	<i>EUSRPWDDTA</i> (on page 361)
	<i>EXCSQLIMM</i> (on page 371)
	<i>EXCSQLSET</i> (on page 377)
	<i>EXCSQLSTT</i> (on page 381)
	<i>OPNQRY</i> (on page 555)
	<i>PRPSQLSTT</i> (on page 636)

**NAME**

SERVER — Server

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD  
**Codepoint** X'1448'  
**Length** \*  
**Class** CLASS  
**Sprcls** MANAGER - Resource Manager

Server (SERVER) Resource Manager is a structured and managed collection of data. The DDM architecture models a server composed of MANAGER objects that contain and organize data for usability and availability.

One or more servers can exist on a single system. For example, an AS/400 is viewed as having a single server; that is, a single collection of files is managed. Alternatively, each of the subsystems of a System/390 running MVS, such as CICS, can be viewed as its own server managing its own collection of data.

Each of these servers has its own identifier and is viewed as a subclass of the class SERVER. This is the basis for negotiation of data connectivity between systems.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'1448'	
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>mgrlvln</b>	1	
<b>mgrdepls</b>	<b>NIL</b>	
<b>vldattls</b>	<b>VALID ATTRIBUTES</b>	
X'0019'	INSTANCE_OF	HELP - Help Text
X'116D'	INSTANCE_OF	SRVNAM - Server Name
X'0045'	INSTANCE_OF	TITLE - Title

**SEE ALSO**

**Semantic** *EXCSAT* (on page 363)  
*EXCSATRD* (on page 369)  
*INHERITANCE* (on page 437)  
*LMTBLKPRC* (on page 475)  
*STRLYR* (on page 890)  
*SUBSETS* (on page 902)

**NAME**

SESDMG — Session Damage Severity Code

**DESCRIPTION (Semantic)****Dictionary** QDDPRMD**Codepoint** X'003F'**Length** \***Class** CONSTANT

Session Damage Severity Code (SESDMG) specifies that damage has occurred to the target server's ability to continue the communications session. It is impossible to continue the current session, but it may be possible to use other available communications sessions.

To recover from session damage, terminate the current session and establish a new session.

---

value	128
-------	-----

**SEE ALSO**

**insvar**            *CMDCHKRM* (on page 173)  
                      *PRCCNVRM* (on page 625)  
                      *RSCLMTRM* (on page 778)  
                      *SVRCOD* (on page 911)

## NAME

SEVERE — Severe Error Severity Code

## DESCRIPTION (Semantic)

**Dictionary** QDDPRMD**Codepoint** X'003A'**Length** \***Class** CONSTANT

Severe Error Severity Code (SEVERE) specifies that a severe error has occurred while executing the command. It was not possible to prevent or backout all changes to objects the command affected. For example, record locks or cursor position may have been lost.

It is possible to send further commands to the affected objects.

---

value	16
-------	----

## SEE ALSO

**insvar**            *AGNPRMRM* (on page 65)  
                       *CMDCHKRM* (on page 173)  
                       *OBJNSPRM* (on page 542)  
                       *PRCCNVRM* (on page 625)  
                       *RSCLMTRM* (on page 778)  
                       *SECCHKRM* (on page 809)  
                       *SVRCOD* (on page 911)

**Semantic**        *DSS* (on page 308)



**NAME**

SNAADDR — SNA Address

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD  
**Codepoint** X'11E9'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

SNA Address (SNAADDR) is an SNA fully qualified network name followed by the SNA LU 6.2 transaction program name of the server.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'11E9'	
luname	INSTANCE_OF LENGTH REQUIRED NOTE	CHRSTRDR - Character String 17  The network identifier and LU name separated by a period and blank padded on the right.
tpname	INSTANCE_OF MAXLEN REQUIRED NOTE	CHRSTRDR - Character String 64  SNA LU 6.2 transaction program name.
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** ACCRDBRM (on page 48)  
 SRVLSRV (on page 876)  
 SYNCLOG (on page 922)  
**Semantic** SRVLSRV (on page 876)

**NAME**

SNASECOVR — LU 6.2 Security Overview

**DESCRIPTION (Semantic)**

**Dictionary** QDDTTRD

**Length** \*

**Class** HELP

The LU 6.2 Security Overview (SNASECOVR) provides an overview of the System Network Architecture (SNA) security mechanism. Normally, the SNA LU 6.2 security mechanism is executed before the DDM target server is started. For more information and details about SNA LU 6.2 security see the *SNA Transaction Programmer's Reference Manual for LU Type 6.2* (GC30-3084, IBM).

**SEE ALSO**

**Semantic** *SECOVR* (on page 819)

**NAME**

SNDPKT — Send Packet

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1805'

**Length** \*

**Class** CODPNT

**Sprcls** COMMAND - Command

Send Packet (SNDPKT) is used by the source server to test its connectivity to the target server. This command can be sent at any time following an EXCSAT. The source server can optionally send a PKTOBJ object along with the command.

The *respktsz* parameter specifies the size of the PKGOBJ object that the target server must return as a normal response, even if the size is 0.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'1805'	
rdbnam	INSTANCE_OF OPTIONAL CMDTRG	RDBNAM - Relational Database Name
respktsz	INSTANCE_OF MINLVL OPTIONAL	RESPKTSZ - Response Packet Size 7
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>cmddta</b>	<b>COMMAND OBJECTS</b>	
X'1C04'	INSTANCE_OF MINLVL MINLEN IGNORABLE OPTIONAL	PKTOBJ - Packet Object 7 0
<b>rpydta</b>	<b>REPLY OBJECTS</b>	
X'1C04'	INSTANCE_OF MINLVL MINLEN IGNORABLE REQUIRED	PKTOBJ - Packet Object 7 0

**SEE ALSO**

**Semantic**

*RESPKTSZ* (on page 763)

**NAME**

SPCVAL — Special Value Attribute

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD  
**Codepoint** X'0036'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

Special Value Attribute (SPCVAL) specifies that the attribute value has special meaning in the context which the attribute is specified.

The attribute value must have attributes compatible with those specified for the term.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'0036'	
<b>value</b>	SPRCLS NOTE	FIELD - A Discrete Unit of Data The attributes for this variable vary according to what is being defined. Also, when BITDR, BITSTRDR, HEXDR, or HEXSTRDR values are specified, they are represented as left-justified in the fewest possible number of whole bytes. For example, 123 would be X'1230'.
	REQUIRED	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *DFTVAL* (on page 276)  
*DGRIOPRL* (on page 279)  
*MAXBLKEXT* (on page 494)  
*MAXRSLCNT* (on page 497)  
*QRYROWNBR* (on page 694)  
*RQSCRR* (on page 772)  
**mgrlvln** *CCSIDMGR* (on page 150)  
*MGRLVL* (on page 506)  
**Semantic** *LVLCMP* (on page 486)

NAME

SPRCLS — Superclass

DESCRIPTION (Semantic)

**Dictionary** QDDPRMD

**Codepoint** X'0037'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Superclass (SPRCLS) String is a class from which variables and commands are inherited by the referencing class.

A superclass makes reference to classes which compose (define) a search list. If a named command cannot be found in a class, its superclass is searched. This continues until a superclass of NIL is reached. New classes can therefore depend on existing classes for common or existing commands. The new class, however, can override an existing command by specifying a new command with the same name.

Similarly, when an instance of a class is created, the instance variables of the superclass chain are included in the new instance. Here, also, a class can override a superclass by assigning the same name to the class data it defines.

When SPRCLS is used as an attribute name in the DDM architecture, the specified value is a term which is the superclass of the variable being described.

<b>clsvar</b>	NIL
<b>insvar</b>	CLASS INSTANCE VARIABLES
length	6
class	X'0037'
value	INSTANCE_OF    CODPNTDR - Codepoint Data Representation REQUIRED
<b>clscmd</b>	NIL
<b>inscmd</b>	NIL

SEE ALSO

- clsvar**            *ENUVAL* (on page 347)
- insvar**            *ARRAY* (on page 101)
- ASSOCIATION* (on page 103)
- ATTLIST* (on page 105)
- CNSVAL* (on page 221)
- DEFINITION* (on page 267)
- DFTVAL* (on page 276)
- MAXVAL* (on page 499)
- MINVAL* (on page 519)
- QLFATT* (on page 664)
- RESERVED* (on page 762)
- SPCVAL* (on page 831)

<b>Semantic</b>	<i>CLASS</i> (on page 158)
<b>sprcls</b>	<i>CLASS</i> (on page 158)

**NAME**

SPVNAM — Supervisor Name

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'115D'

**Length** \*

**Class** CLASS

**Sprcls** MGRNAM - Manager Name

Supervisor Name (SPVNAM) specifies the name of a DDM server's supervisor.

SPVNAM is specified as a null parameter (length = four, and no name value is specified).

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	4	
class	X'115D'	
name	INSTANCE_OF	BYTSTRDR - Byte String
	ENULEN	0
	REQUIRED	
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *EXCSAT* (on page 363)

**vldattls** *SUPERVISOR* (on page 908)



**NAME**

SQL — Structured Query Language

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Length** \***Class** HELP

Structured Query Language (SQL) is the language that accesses data in a relational database (RDB). SQL is defined in *ISO/IEC 9075: 1992, Database Language SQL*.

This term discusses the relationship between the DDM architecture and SQL for distributed RDB processing. Also see the following terms:

RDB	Relational Database (see <i>RDB</i> (on page 718)) This term describes the various objects of an RDB.
RDBOVR	Relational Database Overview (see <i>RDBOVR</i> (on page 740)) This term describes the DDM model of remote RDB processing.
SQLAM	SQL Application Manager (see <i>SQLAM</i> (on page 847)) This term describes how to obtain access to a remote RDB.

**SQL Statements**

Any character string can be transmitted to a remote RDB as an SQL statement. The receiving RDB determines whether it is a syntactically and semantically valid SQL statement and processes it accordingly. This allows programs to use SQL capabilities available from remote RDBs that are not available locally.

SQL statements can be categorized into the following two types:

- Embedded in the source files of application programs written in languages such as C, COBOL, and FORTRAN

Before these source files can be compiled, an appropriate SQL precompiler must generate a modified source file in which each embedded SQL statement is replaced by a comment and a statement in the host language accessing the RDB.

- Typed in from a terminal or built by a program

These are called dynamic SQL statements. Dynamic SQL statements create query programs tailored to specific users and designed for specific needs. They also make it possible to write programs with greater runtime flexibility than embedded SQL statements allow since resource names and SQL statements are varied to meet changing program needs.

All SQL statements can be embedded in programs, but some SQL statements can only be embedded. Examples of SQL statements:

- Embedded or Dynamic
  - Data manipulation functions such as SELECT and DELETE
  - Data definition functions such as CREATE and ALTER
  - Other functions such as COMMIT and SET

- Embedded Only
  - Cursor-related functions such as OPEN and CLOSE
  - Program control functions such as INCLUDE and WHENEVER
  - Dynamic SQL control functions such as PREPARE and EXECUTE
  - The function SET

**Note:** The SET statement appears in both the *Embedded Only* and the *Embedded or Dynamic* categories. This is because some SET statements are *Embedded Only* and some are *Embedded or Dynamic*.

All SQL statements are executed within the context of an application program bound to an RDB package. The embedded SQL statements of the application are bound into permanent sections of the package for execution by reference to their section numbers. Dynamic statements issued without reference to input or output variables can be executed as clauses of the SQL EXECUTE IMMEDIATE statement. All other dynamic statements, however, must be prepared and entered into an existing section of the package by using the SQL PREPARE statement. The SQL EXECUTE statement executes these statements.

### SQL Communications Area

An SQL communications area (SQLCA) is a collection of application program variables that are updated after the execution of every SQL statement. An application program that contains executable SQL statements must provide exactly one SQLCA.

SQLCAs are returned to the application program by the SQLAM for the commands sent to the SQLAM. DDM does not specify how or when this information is assigned to program variables.

See *SQLCARD* (on page 855) for an explanation of how DDM objects carry SQLCA information.

### SQL Descriptor Area

An SQL descriptor area (SQLDA) is a collection of application program variables that are required for the execution of the SQL DESCRIBE statement, and that other statements, such as the OPEN and EXECUTE statements, may use. The meaning of the information in an SQLDA depends on its use. In the DESCRIBE statement, an SQLDA provides information to the host program about a prepared statement. In the OPEN and EXECUTE statements, an SQLDA provides information about host variables. DDM does not specify how or when this information is assigned to program variables.

See *SQLCARD* (on page 855) for a description of how DDM objects carry SQLDA information.

### User Identifiers (USERIDs)

User identifiers (USERIDs) identify users to a server and validate each user's rights to execute SQL statements. A USERID is applied before the execution of every SQL statement.

- For embedded statements:
  1. The USERID of the owner of the RDB package (described in the term PKGOWNID) is checked at bind time to determine which resource objects of the database are accessed by the SQL statements of the program.
  2. The USERID of the user running the SQL program is checked at run time to determine if that user has the right to execute the embedded SQL statements of the named database package.
- For dynamic statements, the USERID of the user running the SQL program is checked at runtime to determine whether the user has the right to access the resource objects of the

database SQL statement identifies.

The DDM architecture requires only that USERIDs be unique within the scope of the remote database. This name has a length of eight bytes and must consist of uppercase letters (A through Z) and numerics (0 through 9). It is transmitted to the remote target server either during conversation initiation or as a parameter on the BGNBND or REBIND command.

The following are assumed:

- The end user must obtain a unique USERID at the remote server and at the RDB.
- An end user might need a different USERID to access each desired server and RDB.

**Note:** USERIDs are also referred to as Authorization Identifiers in some SQL documents.

### Relational Database Names

An RDB name uniquely identifies an instance of an RDB throughout the set of interconnected networks and is associated with a DDM server to which a user accesses by using a specific SNA transaction program name (TPN) at a unique NETID.LUNAME. DDM permits the association of multiple RDB names with a single TPN at a NETID.LUNAME. However, DDM does not specify the mechanism that derives the NETID.LUNAME and TPN pair from the RDB name. Some derivation mechanisms are implementation-specific.

RDB names are carried in the following DDM objects:

PKGNAMECSN	RDB Package Name, Consistency Token, and Section Number (see <i>PKGNAMECSN</i> (on page 596))
PKGNAMECT	RDB Package Name and Consistency Token (see <i>PKGNAMECT</i> (on page 598))
RDBNAME	Relational Database Name (see <i>RDBNAME</i> (on page 736))

### SQL Object Names

SQL statements include references to named objects within an RDB, such as tables, views, and packages (see *RDB* (on page 718) for descriptions of these objects). Each of these names must be unique to allow unambiguous references to it. To allow the distribution of data among a network of cooperating RDBs, each of these objects must have a name that is globally unique across the set of all cooperating RDBs. Users can then uniquely identify an object as the target of an SQL statement. Globally unique names also allow objects to be moved from location to location in a network.

### RDB Object Collection Identifiers

A collection identifier (ID) identifies a unique, user-defined collection of objects contained within an RDB instance. A collection ID can be the object owner's USERID. An installation default mechanism defines the RDB default for collection IDs. The default collection ID can be a USERID.

The term RDBCOLID defines the syntax of RDB object collection IDs. Collection IDs are carried by the following DDM objects:

PKGNAM	RDB Package Name (see <i>PKGNAM</i> (on page 594))
PKGNAMECSN	RDB Package Name, Consistency Token, and Section Number (see <i>PKGNAMECSN</i> (on page 596))
PKGNAMECT	RDB Package Name and Consistency Token (see <i>PKGNAMECT</i> (on page 598))

**Table and View Names**

The globally unique, fully qualified name for a table or view is:

- *rdbnam.rdbcolid.objnam*
  - *rdbnam* is the name of the RDB.
  - *rdbcolid* is the name of a collection of objects in the RDB.
  - *objnam* identifies a table or view. The syntax of object names is defined in the *System Application Architecture Common Programming Interface Database Reference*.
  - The combination of *rdbcolid* and *objnam* uniquely identifies a table or view within the RDB.

**Package Names**

The globally unique, fully qualified name for a package is:

- *rdbnam.rdbcolid.pkgid* with *cnstkn* or *version*
  - *rdbnam* is the RDB name.
  - *rdbcolid* is the RDB collection ID.
  - *pkgid* specifies the name of a package within a collection in the RDB. The term PKGID defines the syntax of package IDs.
  - *cnstkn* specifies a consistency token the RDB checks in all accesses obtained by using the RDB package. Checking this token ensures that the host program and the package remain consistent over time as changes are made to the host program or to the RDB. The term PKGCNSTKN defines consistency tokens.
  - *version* specifies the version attribute of a package. Multiple versions of the SQL program can exist to meet a variety of program development needs. The term VRSNAM defines version names.

To identify a package uniquely, either a version ID or a consistency token must be specified. Version IDs and consistency have a one-to-one ratio.

Package names are carried by the following DDM objects:

PKGNAME	RDB Package Name (see <i>PKGNAME</i> (on page 594))
PKGNAMECSN	RDB Package Name, Consistency Token, and Section Number (see <i>PKGNAMECSN</i> (on page 596))
PKGNAMECT	RDB Package Name and Consistency Token (see <i>PKGNAMECT</i> (on page 598))

**SQL Data Types**

All data stored in an RDB or transferred between application programs and an RDB are defined in terms of SQL data types. These include all user data, SQLCA, and SQLDA data. Since these data types are represented differently (as bit strings) in different servers, it is necessary to convert data as it is transmitted between different servers. For data the target SQLAM receives, the target SQLAM converts the data. For data the source SQLAM receives, the source SQLAM converts the data. The source and target SQLAMs exchange the data type to data representation specifications during ACCRDB command processing. In these specifications, each SQL data type is associated with a specific FD:OCA description of its representation.

**SQL Program Preparation and Execution**

Figure 3-75 (on page 840) illustrates the information flow occurring when a program with embedded SQL statements is prepared for execution against an RDB. This model shows what happens within a single system. Preparation of a program access a remote RDB is similar but operates transparently through the action of the SQL application manager (SQLAM).

Information is input to the process by application interfaces, by defaults, and by being stored in various objects at various stages of the process. In different systems, pieces of information can come from different sources, and not all steps of this process are necessarily performed as illustrated. For example, in some systems the precompilation and bind steps can be combined in a single step while in other systems the precompilation and compilation steps can be combined in a single step. Each systems can implement the SQL program preparation process in the way that best suits its individual processing environment.

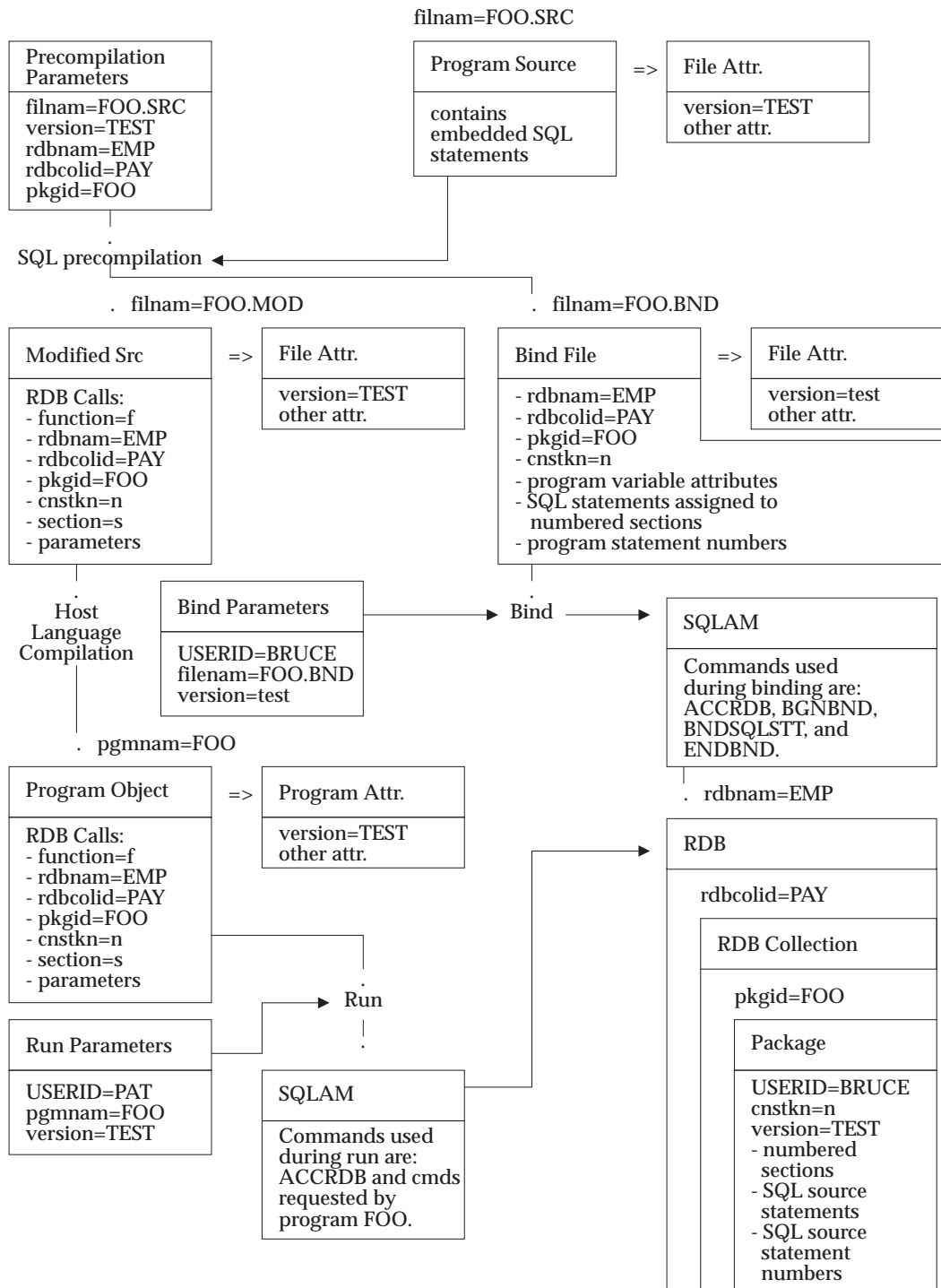


Figure 3-75 Processing Model for Programs with Embedded SQL Statements

### Precompilation

The first step in preparing an application program with embedded SQL statements for execution is precompilation. The input parameters to precompilation are:

1. The name of the file containing the SQL program
2. The version of the file containing the SQL program
3. The name of the RDB the program is requesting
4. The name of the RDB object collection that includes the package
5. The ID of the access package being created in the RDB

The precompiler reads the program source file and produces:

1. A modified source file in which the SQL statements have been replaced by host language comments and calls to the RDB manager.

The modified source file can be virtual in systems that combine precompilation and compilation. The following information is specified in each call to the RDB manager:

- The RDB management function being performed  
This is not the SQL statement; it is the identifier of a local RDB manager function that can be mapped to an SQLAM command.
- The name of the RDB being accessed
- The RDB collection ID that includes the access package
- The ID of the access package
- A consistency token associating the program and the package
- The number of the access package section used in performing the RDB management function
- Depending on the RDB function being performed, additional information can also be passed, including data from the host and from application program variables. In addition, the locations of application program variables that receive answer set data, an SQL communications area, or an SQL descriptor area can be passed.

2. A bind file containing information extracted from the source file.

This information is used in the *bind* step to add an access package to the RDB to meet the specific processing needs of the application program. The bind file can be virtual in systems that combine precompilation and binding.

The bind file consists of:

- The name of the RDB to which the program is to be bound
- The name of the RDB collection identifier to include the access package
- The name or ID of the access package
- The consistency token that the SQL precompiler assigns
- Descriptions of application program variables and structures declared in the program and referenced for input or output by an SQL statement
- Numbered sections containing the text of one or more SQL statements extracted from the source program

The *EXEC SQL* prefix and ";" terminator of each statement are not copied into the bind file, nor are any host language comments embedded in the SQL statement. References to host language variables or structures found in the SQL statement are replaced with one or more ":H" markers. Each single variable is replaced by a single ":H". Each structure is replaced by a sequence of ":H"s, one for each variable in the structure.

At runtime, a DDM command is sent to an SQLAM for each call to the RDB. The command sent to the SQLAM depends on the RDB function being performed which the precompiler from the SQL statement being executed determines:

DDM Command	SQL Statements
ACCRDB	CONNECT
PRPSQLSTT	PREPARE
EXCSQLIMM	EXECUTE IMMEDIATE
OPNQRY	OPEN
CLSQRY	CLOSE
CNTQRY	FETCH
DSCSQLSTT	DESCRIBE STATEMENT
DSCRDBTBL	DESCRIBE TABLE
EXCSQLSTT	EXECUTE (ALTER TABLE, COMMENT ON, CREATE, DELETE, DROP, GRANT, INSERT, LOCK TABLE, SELECT (embedded), REVOKE, UPDATE, SET, LABEL ON)
	Execute any SQL statements that do not include parameter markers or references to host variables such as EXPLAIN.
RDBCMM	COMMIT Only embedded COMMIT statements cause this function call to the RDB. The RDB interrupts and performs dynamic COMMIT statements.
RDBRLLBCK	ROLLBACK Only embedded ROLLBACK statements cause this function call to the RDB. The RDB interrupts and performs dynamic ROLLBACK statements.
No command	DECLARE CURSOR, DECLARE STATEMENT, DECLARE TABLE, INCLUDE, RELEASE, SET CONNECTION, WHENEVER, and local SQL statements.  An SQL statement completely processed by the precompiler is defined as a <i>local SQL statement</i> .

The rules the precompiler follows in assigning section numbers to SQL statements are in the DRDA Reference.

### Host Language Compilation

The second step in program preparation is to compile the modified source program by using the appropriate host language compiler. After compilation (and link-editing in environments that require that step), the program is ready for execution, but it is not ready to access the RDB. The compiled program first must be bound to the RDB.



## Binding

The third step of program preparation is to bind the program to the RDB. This is done by sending bind commands (BGNBND, BNDSQLSTT, and ENDBND; or REBIND) to an SQLAM that has access to the RDB. If the RDB is located on a remote system, the SQLAM handles all communications with the remote RDB. See *RDBOVR* (on page 740) for information about binding.

The input parameters to the bind process are:

1. The USERID of the user performing the bind
2. The name of the bind file
3. The version of the bind file

The bind process reads the bind file and creates a package in the RDB. Not all of the SQL statements in the bind file are sent to the target SQLAM. The SQL statements not sent to the target SQLAM during the bind process are: INCLUDE, WHENEVER, PREPARE, EXECUTE, EXECUTE IMMEDIATE, DESCRIBE, DESCRIBE TABLE, OPEN, FETCH, CLOSE, COMMIT, ROLLBACK, BEGIN DECLARE SECTION, END DECLARE SECTION, and other local SQL statements.

The package consists of numbered sections containing access algorithms for the corresponding SQL statements in the bound program, reserved sections for dynamic SQL statements, and a copy of the bind processing options. The text of the SQL statements is also retained in the package to allow the RDB to perform automatic rebinds if the changes to the RDB require them. The attributes of the package are:

- The USERID of the user who is the owner of the package
- The consistency token of the program and the package
- The version of the program and the package

Two additional SQLAM commands are available for managing packages:

1. The drop package (DRPPKG) command can remove a package from an RDB.
2. The rebind (REBIND) command can be used to recreate a package in the RDB so that it takes advantage of recent changes to the database, such as the definition of new indexes. A bind file is not needed because the existing package contains all the required information.

## Program Execution

Finally, the executable program version can be run. DDM does not specify how the program is invoked or the environment in which it is run. The input parameters to program execution are:

1. The USERID of the user running the program  
This USERID need not be the same as the USERID that bound the program to the RDB. It validates the requester's right to execute the program and to access the RDB.
2. The name of the program being executed
3. The version of the program being executed

During execution, the program's *RDB Calls* shown in Figure 3-75 (on page 840) cause access commands to be sent to an SQLAM instance specifying the number of a package section used in accessing the RDB. If the RDB is located on a remote system, the SQLAM handles all communications with the remote RDB. For more information, see *RDBOVR* (on page 740).

## SEE ALSO

<b>cmddda</b>	<i>BNDSQLSTT</i> (on page 136) <i>DSCRDBTBL</i> (on page 300) <i>EXCSQLIMM</i> (on page 371) <i>EXCSQLSET</i> (on page 377) <i>EXCSQLSTT</i> (on page 381) <i>OPNQRY</i> (on page 555) <i>PRPSQLSTT</i> (on page 636)
<b>cmdrpy</b>	<i>DSCSQLSTT</i> (on page 304)
<b>inscmd</b>	<i>SQLAM</i> (on page 847)
<b>insvar</b>	<i>BNDEXPOPT</i> (on page 129) <i>DGRIOPRL</i> (on page 279) <i>DSCERRCD</i> (on page 296) <i>DSCSQLSTT</i> (on page 304) <i>MGRLVL</i> (on page 506) <i>PKGATHRUL</i> (on page 581) <i>PRPSQLSTT</i> (on page 636) <i>RDBACCCL</i> (on page 723) <i>STTSTRDEL</i> (on page 899) <i>TYPDEFNAM</i> (on page 1027)
<b>rpydta</b>	<i>BNDSQLSTT</i> (on page 136) <i>CLSQRY</i> (on page 165) <i>CNTQRY</i> (on page 222) <i>DRPPKG</i> (on page 293) <i>DSCRDBTBL</i> (on page 300) <i>DSCSQLSTT</i> (on page 304) <i>ENDBND</i> (on page 336) <i>EXCSQLIMM</i> (on page 371) <i>EXCSQLSET</i> (on page 377) <i>EXCSQLSTT</i> (on page 381) <i>OPNQRY</i> (on page 555) <i>PRPSQLSTT</i> (on page 636) <i>RDBCMM</i> (on page 728) <i>RDBRLLBCK</i> (on page 745) <i>REBIND</i> (on page 753)
<b>Semantic</b>	<i>ACCRDBRM</i> (on page 48) <i>AGENT</i> (on page 61) <i>APPSRCCD</i> (on page 79) <i>BGNBND</i> (on page 110) <i>BNDCHKEXS</i> (on page 125) <i>BNDCHKONL</i> (on page 126) <i>BNDERRALW</i> (on page 128) <i>BNDNERALW</i> (on page 132) <i>BNDSQLSTT</i> (on page 136) <i>BNDSTTASM</i> (on page 141) <i>DECDELPRD</i> (on page 265) <i>DFTPKG</i> (on page 273) <i>DGRIOPRL</i> (on page 279) <i>DSCSQLSTT</i> (on page 304)

*DTAOVR* (on page 321)  
*ENDBND* (on page 336)  
*EURTIMFMT* (on page 354)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSET* (on page 377)  
*EXCSQLSTT* (on page 381)  
*EXPALL* (on page 393)  
*EXPNON* (on page 394)  
*FDOCA* (on page 404)  
*ISODATFMT* (on page 451)  
*ISOLVLALL* (on page 452)  
*ISOLVLCHG* (on page 453)  
*ISOLVLCS* (on page 454)  
*ISOLVLNC* (on page 455)  
*ISOLVLR* (on page 456)  
*ISOTIMFMT* (on page 457)  
*JISDATFMT* (on page 458)  
*JISTIMFMT* (on page 459)  
*MGROVR* (on page 514)  
*OPNQRYSM* (on page 566)  
*OUTEXP* (on page 571)  
*OUTOVR* (on page 572)  
*OWNER* (on page 577)  
*PKGATHRUL* (on page 581)  
*PKGDFTCST* (on page 590)  
*PKGISOLVL* (on page 592)  
*PKGOWNID* (on page 600)  
*PRPSQLSTT* (on page 636)  
*QRYBLKCTL* (on page 672)  
*QRYBLKSZ* (on page 678)  
*RDB* (on page 718)  
*RDBACCCL* (on page 723)  
*RDBOVR* (on page 740)  
*RDBRISOPT* (on page 750)  
*REBIND* (on page 753)  
*REQUESTER* (on page 760)  
*RTNSQLDA* (on page 795)  
*SQLAM* (on page 847)  
*SQLDTA* (on page 860)  
*SQLDTARD* (on page 862)  
*SQLSTT* (on page 868)  
*SQLSTTNBR* (on page 870)  
*SQLSTTVRB* (on page 871)  
*STRDELAP* (on page 886)  
*STRDELQ* (on page 887)  
*STTASMEUI* (on page 893)  
*STTDECDEL* (on page 897)  
*STTSCCLS* (on page 898)  
*STTSTRDEL* (on page 899)  
*STTTIMFMT* (on page 900)  
*TCPSRCCD* (on page 1007)  
*TYPDEF* (on page 1025)

*TYPSQLDA* (on page 1034)  
*USADATFMT* (on page 1041)  
*USATIMFMT* (on page 1042)

**NAME**

SQLAM — SQL Application Manager

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

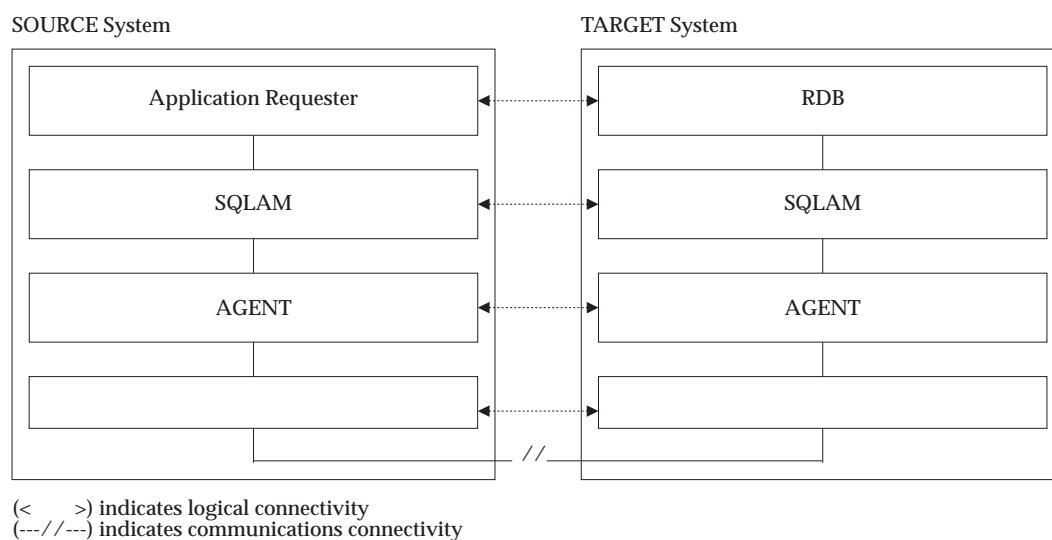
**Codepoint** X'2407'

**Length** \*

**Class** CLASS

**Sprcls** MANAGER - Resource Manager

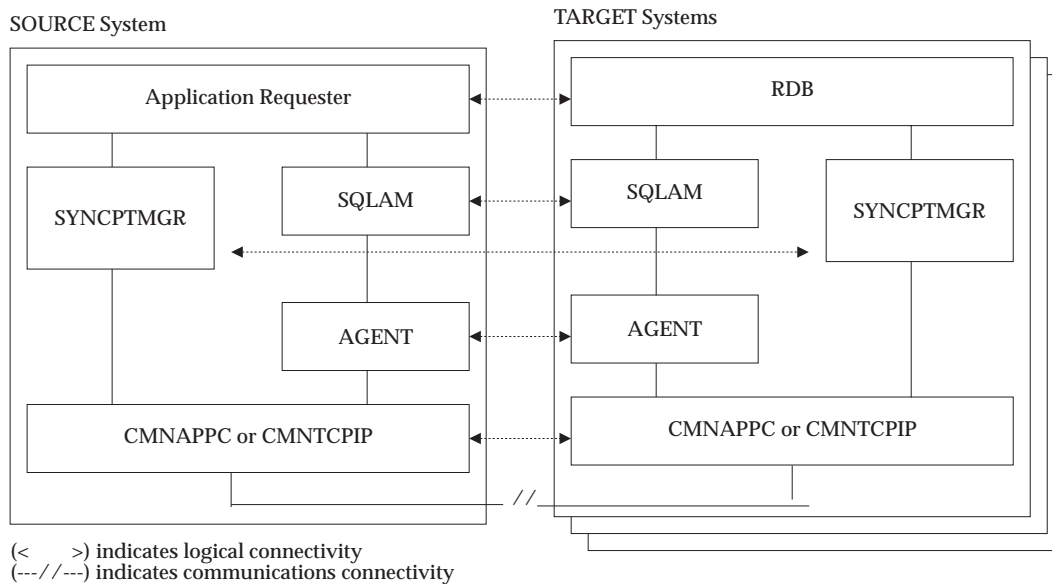
SQL Application Manager (SQLAM) provides a consistent method of requesting SQL services from a relational database (RDB) for a single application requester. In doing so, it also provides for the transparent distribution of SQL statement processing to remote RDBs as illustrated in Figure 3-76.



**Figure 3-76** Application Requester and a Remote RDB

Figure 3-77 (on page 848) illustrates the two-phase commit sync point manager (SYNCPTMGR). If using DDM sync point commands, the sync point operation is supported by the sync point manager at Level 5 and the resynchronization manager (RSYNCMGR) at Level 5 when the sync point operation fails. Communications sync point (CMNSYNCPT) support is provided by the sync point manager at Level 4 using LU 6.2 sync point support. The SYNCPTMGRs logically connect to each other. The source SYNCPTMGR is a coordinator for the local resources and the target system. The target SYNCPTMGR protects the target's resources and might also act as a coordinator if recoverable resources are on another target system beyond this target. Under two-phase commit processing, resources on source and multiple targets can be protected concurrently.

See *RDBOVR* (on page 740) and *SQL* (on page 835) for overviews.



**Figure 3-77** Application Requester, Remote RDBs, and Two-Phase Commit

### Data Conversions

All SQL data is described in terms of the *data types* SQL defines. Data types include user data and SQL-defined data such as SQL statements, the SQL Communications Area (SQLCA), and the SQL Descriptor Area (SQLDA). However, SQL data types are represented differently in different systems. The target SQLAM performs conversions of SQL data type representations as data flows from the source system to the target system, and the source SQLAM performs conversions of SQL data type representations as data flows from the target system to the source system.

### Controlling SQL Type Definitions

When a source SQLAM is created, it adopts the SQL TYPDEF of the requesting program. However, this SQL TYPDEF can be overridden when the preferred SQL TYPDEF of a target SQLAM is received in response to an ACCRDB command. If the returned SQL TYPDEF is the same as the SQL TYPDEF of the requesting program, no data conversions are required. Otherwise, the source SQLAM must do one of the following:

- Terminate the access of the remote RDB if it cannot perform the necessary data conversions.
- Adopt the SQL TYPDEF of the target SQLAM and prepare to perform the necessary conversions.

The SQL TYPDEF attribute of a target SQLAM is set when the target SQLAM is created. If the SQL TYPDEF of the accessed RDB is the same as the SQL TYPDEF specified on the ACCRDB command, then no data conversions are required. Otherwise, the target SQLAM must do one of the following:

- Adopt the SQL TYPDEF of the RDB and return this SQL TYPDEF to the source SQLAM requesting that the source SQLAM perform the necessary data conversions.
- Adopt the SQL TYPDEF specified on the ACCRDB command and perform the necessary data conversions.

### Query Processing

The source SQLAM uses the open query (OPNQRY), continue query (CNTQRY), and close query (CLSQR) commands to efficiently handle SQL statements that can return large amounts of answer set data. This is called *query processing*. The reply data consists of a set of FD:OCA descriptor objects that describe the answer set data and a set of data objects that carry the answer set data. The answer set data consists of a set of rows each containing a set of output data values.

The source SQLAM issues the OPNQRY command in response to the application issuing the SQL *OPEN* statement. The target SQLAM responds by issuing (local interface) requests to the RDB to:

1. Obtain a description of the query output.
2. Open the query by package section number and provide the input variable values the source SQLAM transmits.
3. Fetch query output rows to fill one or more blocks with answer data.
4. Return an FD:OCA description of the answer set data.
5. Return the answer set data.

The amount of answer set data the SQLAM returns in response to the OPNQRY and CNTQRY commands is determined by the query protocol being used. *OPNQRY* (on page 555) contains more information about the query protocols. The query protocols are:

- Fixed row, which returns a specified number of rows of answer data

The CNTQRY command obtains the answer set. This protocol is described in the term FIXROWPRC.

- Limited block, which returns one or more rows of answer set data

In this case, the source SQLAM uses the OPNQRY and CNTQRY commands to obtain rows of answer set data, as needed. The source SQLAM extracts individual rows from the blocks it receives in response to the OPNQRY and CNTQRY commands. This protocol is described in the term LMTBLKPRC.

The source SQLAM uses the CLSQR command to terminate a query at any point at which the query is suspended. For non-scrollable cursors, it is not necessary for the source SQLAM to issue the CLSQR command if the target SQLAM returned all of the answer set data, and the ENDQRYRM has been received. For scrollable cursors, the target SQLAM will never send an ENDQRYRM. The source SQLAM must always issue a CLSQR.

Multiple queries can be in process at the same time. This allows the source SQLAM to respond to requests to fetch rows from different RDB cursors.

### Update Privileges

The source SQLAM is responsible for enforcing update privileges at the target SQLAMs:

- If the application is not using the services of a SYNCPTMGR in the unit of work (UOW), updating is restricted to one RDB.
- If the application is using the services of a SYNCPTMGR and the remote RDBs also use SYNCPTMGRs, updating is allowed for those RDBs. The SNA LU 6.2 two-phase commit protocols or SYNCPTMGR Level 5 must be used.

- Target SQLAMs must send the RDBUPDRM after the first recoverable update is applied to the target RDB within each unit of work.

See the DRDA Reference for more information about enforcing update privileges.

### Manager-Level Compatibility

Table 3-15 illustrates the function of the SQLAM as it has grown and changed through the levels of DDM architecture.

**Table 3-15** SQL Application Manager-Level Compatibility

<b>DDM Levels</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
SQLAM			3	4	5
<i>Commands</i>					
ACCRDB			3	4	4
BGNBND			3	4	5
BNDSQLSTT			3	4	4
CLSQRY			3	3	3
CNTQRY			3	3	5
DRPPKG			3	4	4
DSCRDBTBL			3	4	4
DSCSQLSTT			3	4	4
ENDBND			3	4	4
EXCSQLIMM			3	4	4
EXCSQLSTT			3	4	5
INTRDBRQS			3	3	3
OPNQRY			3	4	5
PRPSQLSTT			3	3	3
RDBCMM			3	4	4
RDBRLLBCK			3	4	4
REBIND			3	4	5
<i>Manager Dependencies</i>					
AGENT			3	4	4
RDB			3	3	3
SECMGR <sup>16</sup>			1	1	5

16. The SECMGR can be at DDM Level 1 or DDM Level 5.



<b>clsvar</b>	NIL	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'2407'	
typdef	INSTANCE_OF	TYPDEF - Data Type to Data Representation Definition
<b>clscmd</b>	<b>CLASS COMMANDS</b>	
X'2001'	INSTANCE_OF REQUIRED	ACCRDB - Access RDB
<b>inscmd</b>	<b>INSTANCE COMMANDS</b>	
X'2002'	INSTANCE_OF REQUIRED	BGNBND - Begin Binding a Package to an RDB
X'2004'	INSTANCE_OF REQUIRED	BNDSQLSTT - Bind SQL Statement to an RDB Package
X'2005'	INSTANCE_OF REQUIRED	CLSQRV - Close Query
X'2006'	INSTANCE_OF REQUIRED	CNTQRY - Continue Query
X'2007'	INSTANCE_OF REQUIRED	DRPPKG - Drop RDB Package
X'2012'	INSTANCE_OF OPTIONAL	DSCRDBTBL - Describe RDB Table
X'2008'	INSTANCE_OF REQUIRED	DSCSQLSTT - Describe SQL Statement
X'200A'	INSTANCE_OF REQUIRED	EXCSQLIMM - Execute Immediate SQL Statement
X'200B'	INSTANCE_OF REQUIRED	EXCSQLSTT - Execute SQL Statement
X'2003'	INSTANCE_OF OPTIONAL	INTRDBRQS - Interrupt RDB Request
X'200C'	INSTANCE_OF REQUIRED	OPNQRY - Open Query
X'200D'	INSTANCE_OF REQUIRED	PRPSQLSTT - Prepare SQL Statement
X'200E'	INSTANCE_OF REQUIRED	RDBCMM - RDB Commit Unit of Work
X'200F'	INSTANCE_OF REQUIRED	RDBRLLBCK - RDB Rollback Unit of Work
X'2010'	INSTANCE_OF OPTIONAL	REBIND - Rebind an Existing RDB Package
<b>mgrlvl</b>	5	
<b>mgrdepls</b>	<b>MANAGER DEPENDENCY LIST</b>	
X'1403'	INSTANCE_OF MGRVLN NOTE	AGENT - Agent 3 The agent must be able to route commands based on RDBNAM command targets.

X'240F'	INSTANCE_OF MGRLVLN NOTE	RDB - Relational Database 3 The RDB manager stores all persistent data the SQLAM uses or reads.
X'1440'	INSTANCE_OF NOTE  MGRLVLN NOTE MINLVL	SECMGR - Security Manager The security manager validates that the requester is authorized to use the RDB manager. It does not validate the requester's authorization to the objects stored in the RDB manager. 5 The SECMGR can be at DDM Level 1 or DDM Level 5. 5
<b>vldattls</b>		<b>VALID ATTRIBUTES</b>
X'0019'	INSTANCE_OF	HELP - Help Text
X'1452'	INSTANCE_OF	MGRNAM - Manager Name
X'0045'	INSTANCE_OF	TITLE - Title

**SEE ALSO**

<b>cmdrpy</b>	<i>ACCRDB</i> (on page 42) <i>CNTQRY</i> (on page 222)
<b>insvar</b>	<i>ACCRDB</i> (on page 42) <i>DEPERRC</i> (on page 270) <i>MAXBLKEXT</i> (on page 494) <i>MAXRSLCNT</i> (on page 497) <i>MGRLVL</i> (on page 506) <i>OUTEXP</i> (on page 571) <i>RDBACCCL</i> (on page 723)
<b>Semantic</b>	<i>ABNUOWRM</i> (on page 39) <i>ACCRDB</i> (on page 42) <i>ACCRDBRM</i> (on page 48) <i>AGENT</i> (on page 61) <i>APPCMNFL</i> (on page 68) <i>BGNBND</i> (on page 110) <i>BNDEXPOPT</i> (on page 129) <i>BNDSQLSTT</i> (on page 136) <i>CLSQR</i> (on page 165) <i>CMDCMPRM</i> (on page 175) <i>CMNAPPC</i> (on page 184) <i>CMNSYNCPT</i> (on page 202) <i>CMNTCPIP</i> (on page 214) <i>CNTQRY</i> (on page 222) <i>DIAGLVL</i> (on page 284) <i>DRPPKG</i> (on page 293) <i>DSCRDBTBL</i> (on page 300) <i>DSCSQLSTT</i> (on page 304) <i>ENDBND</i> (on page 336) <i>EXCSQLIMM</i> (on page 371) <i>EXCSQLSET</i> (on page 377) <i>EXCSQLSTT</i> (on page 381)

*EXPALL* (on page 393)  
*EXPNON* (on page 394)  
*EXPYES* (on page 396)  
*FIXROWPRC* (on page 417)  
*INHERITANCE* (on page 437)  
*INTRDDBRQS* (on page 445)  
*INTTKNRM* (on page 448)  
*LMTBLKPRC* (on page 475)  
*MANAGER* (on page 491)  
*MAXBLKEXT* (on page 494)  
*OPNQRY* (on page 555)  
*OUTEXP* (on page 571)  
*PKGOWNID* (on page 600)  
*PRPSQLSTT* (on page 636)  
*QRYBLKCTL* (on page 672)  
*QRYBLKSZ* (on page 678)  
*QRYPRCTYP* (on page 692)  
*QRYROWSET* (on page 697)  
*RDBAFLRM* (on page 725)  
*RDBCMM* (on page 728)  
*RDBINTTKN* (on page 733)  
*RDBOVR* (on page 740)  
*RDBRLLBCK* (on page 745)  
*REBIND* (on page 753)  
*RSCLMTRM* (on page 778)  
*RSLSETFLG* (on page 783)  
*RTNSQLDA* (on page 795)  
*SQL* (on page 835)  
*SUBSETS* (on page 902)  
*SYNCMNBK* (on page 924)  
*SYNCMNFL* (on page 928)  
*SYNCPTMGR* (on page 939)  
*TCPCMNFL* (on page 992)  
*TYPSQLDA* (on page 1034)

## NAME

SQLATTR — SQL Statement Attributes

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD**Codepoint** X'2450'**Length** \***Class** CLASS**Sprcls** FDOTA - FD:OCA Data

SQL Statement Attributes (SQLATTR) specifies the SQL statement attributes being prepared.<sup>17</sup> The SQL language standard or product-unique extensions, not DDM architecture, defines the syntax and semantics of the SQL statement attributes.

With certain SECMECs, the DSS carrier containing the SQLATTR object must be encrypted. (See *EDTASECOVR* (on page 323) for details.)

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'2450'	
<b>sqlattr</b>	INSTANCE_OF	BYTSTRDR - Byte String
	OPTIONAL	
	MINLVL	7
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

## SEE ALSO

**cmdtda** *PRPSQLSTT* (on page 636)**Semantic** *EDTASECOVR* (on page 323)*PRPSQLSTT* (on page 636)

17. The FD:OCA description for the data value of this object is described in the DRDA Reference.

**NAME**

SQLCARD — SQL Communications Area Reply Data

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2408'  
**Length** \*  
**Class** CLASS  
**Sprcls** FDOTA - FD:OCA Data

SQL Communications Area Reply Data (SQLCARD) is a byte string that specifies information about conditions detected during relational database (RDB) processing.<sup>18</sup> An SQLCARD contains an SQLCA.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
length	*
class	X'2408'
value	INSTANCE_OF BYTSTRDR - Byte String REQUIRED
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO**

**insvar** *DIAGLVL* (on page 284)  
**rpydta** *ACCRDB* (on page 42)  
*BGNBND* (on page 110)  
*BNDSQLSTT* (on page 136)  
*CLSQR*Y (on page 165)  
*CNTQR*Y (on page 222)  
*DRPPKG* (on page 293)  
*DSCRDBTBL* (on page 300)  
*DSCSQLSTT* (on page 304)  
*ENDATMCHN* (on page 333)  
*ENDBND* (on page 336)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSET* (on page 377)  
*EXCSQLSTT* (on page 381)  
*OPNQRY* (on page 555)  
*PRPSQLSTT* (on page 636)  
*RDBCMM* (on page 728)  
*RDBRLLBCK* (on page 745)  
*REBIND* (on page 753)

<sup>18</sup> The FD:OCA description for the data value of this object is described in the DRDA Reference.

**Semantic**

- ABNUOWRM* (on page 39)
- ACCRDB* (on page 42)
- BGNATMCHN* (on page 107)
- BGNBND* (on page 110)
- BGNBNDRM* (on page 117)
- BNDNERALW* (on page 132)
- BNDSQLSTT* (on page 136)
- BNDSTTASM* (on page 141)
- CLSQR*Y (on page 165)
- CNTQR*Y (on page 222)
- DIAGLVL* (on page 284)
- DRPPKG* (on page 293)
- DSCRDBTBL* (on page 300)
- DSCSQLSTT* (on page 304)
- ENDATMCHN* (on page 333)
- ENDBND* (on page 336)
- ENDQR*YRM (on page 342)
- EXCSQLIMM* (on page 371)
- EXCSQLSET* (on page 377)
- EXCSQLSTT* (on page 381)
- FIXROWPRC* (on page 417)
- LMTBLKPRC* (on page 475)
- OPNQFLRM* (on page 554)
- OPNQRY* (on page 555)
- PKGRPLNA* (on page 603)
- PRPSQLSTT* (on page 636)
- QRYBLK* (on page 670)
- RDBAFLRM* (on page 725)
- RDBCMM* (on page 728)
- RDBRLLBCK* (on page 745)
- REBIND* (on page 753)
- RSCLMTRM* (on page 778)
- RSLSETRM* (on page 785)
- SQLERRRM* (on page 864)
- SYNCPTOV* (on page 944)

**NAME**

SQLCINRD — SQL Result Set Column Information Reply Data

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'240B'  
**Length** \*  
**Class** CLASS  
**Sprcls** FDOTA - FD:OCA Data

SQL Result Set Column Information Reply Data (SQLCINRD) is a byte string that specifies information about columns for a result set returned as reply data in the response to an EXCSQLSTT command that invokes a stored procedure.<sup>19</sup>

With certain SECMECs, the DSS carrier containing the SQLCINRD object must be encrypted. (See *EDTASECOVR* (on page 323) for details.)

<b>clsvar</b>	NIL
<b>insvar</b>	CLASS INSTANCE VARIABLES
<b>length</b>	*
<b>class</b>	X'240B'
<b>value</b>	INSTANCE_OF BYTSTRDR - Byte String REQUIRED
<b>clscmd</b>	NIL
<b>inscmd</b>	NIL

**SEE ALSO**

**rpydta** *EXCSQLSTT* (on page 381)  
**Semantic** *EDTASECOVR* (on page 323)  
*EXCSQLSTT* (on page 381)  
*LMTBLKPRC* (on page 475)  
*RSLSETFLG* (on page 783)  
*SQLCINRD*

<sup>19</sup> The FD:OCA description for the data value of this object is described in the DRDA Reference.

**NAME**

SQLCSRHLD — Hold Cursor Position

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'211F'  
**Length** \*  
**Class** CLASS  
**Sprcls** BOOLEAN - Logical Value

Hold Cursor Position (SQLCSRHLD) Boolean State indicates whether the requester specified the HOLD option on the SQL DECLARE CURSOR statement. When the HOLD option is specified, the cursor is not closed upon execution of a commit operation.

The value TRUE indicates that the requester specifies the HOLD option.

The value FALSE indicates that the requester is not specifying the HOLD option.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	5	
class	X'211F'	
value	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	LENGTH	2
	ENUVAL	X'F1' - TRUE - True State
	NOTE	Specified HOLD option.
	ENUVAL	X'F0' - FALSE - False State
	NOTE	Did not specify HOLD option.
	DFTVAL	X'F0' - FALSE - False State
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** OPNQRYRM (on page 566)



**NAME**

SQLDARD — SQLDA Reply Data

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2411'

**Length** \*

**Class** CLASS

**Sprcls** FDODTA - FD:OCA Data

SQLDA Reply Data (SQLDARD) contains an SQLCA followed by an SQLDA.<sup>20</sup>

With certain SECMECs, the DSS carrier containing the SQLDA Reply Data object must be encrypted. (See *EDTASECOVR* (on page 323) for details.)

<b>clsvar</b>	NIL
<b>insvar</b>	CLASS INSTANCE VARIABLES
<b>length</b>	*
<b>class</b>	X'2411'
<b>value</b>	INSTANCE_OF BYTSTRDR - Byte String REQUIRED
<b>clscmd</b>	NIL
<b>inscmd</b>	NIL

**SEE ALSO**

**rpydta** *DSCRDBTBL* (on page 300)  
*DSCSQLSTT* (on page 304)  
*PRPSQLSTT* (on page 636)

**Semantic** *DHENC* (on page 280)  
*DSCRDBTBL* (on page 300)  
*DSCSQLSTT* (on page 304)  
*EDTASECOVR* (on page 323)  
*EUSRIDDTA* (on page 355)  
*EUSRPWDDTA* (on page 361)  
*EXCSQLSTT* (on page 381)  
*OPNQRY* (on page 555)  
*PRPSQLSTT* (on page 636)  
*TYPSQLDA* (on page 1034)

<sup>20</sup>. The FD:OCA description for the data value of this object is described in the DRDA Reference.

NAME

SQLDTA — SQL Program Variable Data

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'2412'  
**Length** \*  
**Class** CLASS  
**Sprcls** FDOOBJ - FD:OCA Object

SQL Program Variable Data (SQLDTA) consists of input data to an SQL statement that a relational database (RDB) is executing. It also includes a description of the data.<sup>21</sup>

With certain SECMECs, the DSS carrier containing SQLDTA FD:OCA objects must be encrypted. The FD:OCA objects include the FD:OCA Data Descriptor objects and the FD:OCA Data objects. (See *EDTASECOVR* (on page 323) for details.)

DSS Carrier: OBJDSS

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'2412'	
fdodsc	INSTANCE_OF REQUIRED REPEATABLE	FDODSC - FD:OCA Data Descriptor
fdodta	INSTANCE_OF OPTIONAL REPEATABLE	FDODTA - FD:OCA Data
fdoext	INSTANCE_OF OPTIONAL REPEATABLE MINLVL	FDOEXT - FD:OCA Extent Data 7
fdooff	INSTANCE_OF OPTIONAL REPEATABLE MINLVL	FDOOFF - FD:OCA Offset Data 7
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

21. The FD:OCA description for the data value of this object is described in the DRDA Reference.

**SEE ALSO**

- cmdtta**            *EXCSQLSTT* (on page 381)  
                      *OPNQRY* (on page 555)
  
- Semantic**        *DHENC* (on page 280)  
                      *EDTASECOVR* (on page 323)  
                      *EUSRIDDTA* (on page 355)  
                      *EUSRPWDDTA* (on page 361)  
                      *EXCSQLSTT* (on page 381)  
                      *EXTDTA* (on page 397)  
                      *EXTDTAOVR* (on page 399)  
                      *OPNQRY* (on page 555)

**NAME**

SQLDTARD — SQL Data Reply Data

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2413'  
**Length** \*  
**Class** CLASS  
**Sprcls** FDOOBJ - FD:OCA Object

SQL Data Reply Data (SQLDTARD) consists of output data from the relational database (RDB) processing of an SQL statement. It also includes a description of the data.<sup>22</sup>

With certain SECMECs, the DSS carried containing SQLDTARD FD:OCA objects must be encrypted. The FD:OCA objects include the FD:OCA Data Descriptor objects and the FD:OCA Data objects. (See *EDTASECOVR* (on page 323) for details.)

**DSS Carrier: OBJDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'2413'	
<b>fdodsc</b>	INSTANCE_OF REQUIRED REPEATABLE	FDODSC - FD:OCA Data Descriptor
<b>fdodta</b>	INSTANCE_OF OPTIONAL REPEATABLE	FDODTA - FD:OCA Data
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *OUTEXP* (on page 571)  
**rpydta** *EXCSQLSTT* (on page 381)  
**Semantic** *DHENC* (on page 280)  
*EDTASECOVR* (on page 323)  
*EUSRIDDTA* (on page 355)  
*EUSRPWDDTA* (on page 361)  
*EXCSQLSTT* (on page 381)  
*EXTDTA* (on page 397)  
*EXTDTAOVR* (on page 399)  
*LMTBLKPRC* (on page 475)

<sup>22</sup>. The FD:OCA description for the data value of this object is described in the DRDA Reference.

*OUTEXP* (on page 571)  
*RSLSETRM* (on page 785)

**NAME**

SQLERRRM — SQL Error Condition

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2213'  
**Length** \*  
**Class** CLASS  
**Sprcls** RPYMSG - Reply Message

SQL Error Condition (SQLERRRM) Reply Message indicates that an SQL error has occurred. It may be sent even though no reply message precedes the SQLCARD object that is the normal response to a command when an exception condition occurs.

The SQLERRRM is also used when a BNDSQLSTT command is terminated by an INTRDBRQS command.

The SQLERRRM can also be used to terminate the processing of an atomic chain of EXCSQLSTT commands.

This reply message must precede an SQLCARD object.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'2213'	
<b>rdbnam</b>	INSTANCE_OF OPTIONAL	RDBNAM - Relational Database Name
<b>srvdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code  8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**cmdrpy** *BGNATMCHN* (on page 107)  
*BNDSQLSTT* (on page 136)  
*DSCRDBTBL* (on page 300)  
*DSCSQLSTT* (on page 304)  
*ENDATMCHN* (on page 333)  
*ENDBND* (on page 336)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSET* (on page 377)  
*EXCSQLSTT* (on page 381)  
*PRPSQLSTT* (on page 636)  
*REBIND* (on page 753)

**Semantic**      *BNDSQLSTT* (on page 136)  
*DSCRDBTBL* (on page 300)  
*DSCSQLSTT* (on page 304)  
*EXCSQLSET* (on page 377)  
*PRPSQLSTT* (on page 636)

**NAME**

SQLOBJNAM — SQL Object Name

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'243E'  
**Length** \*  
**Class** CLASS  
**Sprcls** FDODTA - FD:OCA Data

SQL Object Name (SQLOBJNAM) is a byte string that specifies the name of a relational database (RDB) table, collection, index or package.<sup>23</sup> The name can be a one-part, two-part, or three-part RDB object name.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'243E'	
value	INSTANCE_OF MAXLEN REQUIRED	BYTSTRDR - Byte String 256
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**cmdtda** *DSCRDBTBL* (on page 300)  
**Semantic** *DSCRDBTBL* (on page 300)

<sup>23</sup> The FD:OCA description for the data value of this object is described in the DRDA Reference.



**NAME**

SQLRSLRD — SQL Result Set Reply Data

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'240E'

**Length** \*

**Class** CLASS

**Sprcls** FDOTA - FD:OCA Data

SQL Result Set Reply Data (SQLRSLRD) is a byte string that specifies information about result sets returned as reply data in the response to an EXCSQLSTT command that invokes a stored procedure.<sup>24</sup>

With certain SECMECs, the DSS carrier containing the SQLRSLRD object must be encrypted. (See *EDTASECOVR* (on page 323) for details.)

<b>clsvar</b>	NIL
<b>insvar</b>	CLASS INSTANCE VARIABLES
<b>length</b>	*
<b>class</b>	X'240E'
<b>value</b>	INSTANCE_OF BYTSTRDR - Byte String REQUIRED
<b>clscmd</b>	NIL
<b>inscmd</b>	NIL

**SEE ALSO**

- rpydta** *EXCSQLSTT* (on page 381)
- Semantic** *EDTASECOVR* (on page 323)  
*EXCSQLSTT* (on page 381)  
*LMTBLKPRC* (on page 475)

<sup>24</sup> The FD:OCA description for the data value of this object is described in the DRDA Reference.

## NAME

SQLSTT — SQL Statement

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD**Codepoint** X'2414'**Length** \***Class** CLASS**Sprcls** FDOTA - FD:OCA Data

SQL Statement (SQLSTT) specifies that an SQL statement is being either executed or bound into an RDB package.<sup>25</sup>

The SQL language standard or product unique extensions, not the DDM architecture, defines the syntax and semantics of the SQL statement.

With certain SECMECs, the DSS carrier containing SQL statements must be encrypted. (See *EDTASECOVR* (on page 323) for details.)

**DSS Carrier: OBJDSS**

<b>clsvar</b>	NIL
<b>insvar</b>	CLASS INSTANCE VARIABLES
<b>length</b>	*
<b>class</b>	X'2414'
<b>value</b>	INSTANCE_OF BYTSTRDR - Byte String REQUIRED
<b>clscmd</b>	NIL
<b>inscmd</b>	NIL

## SEE ALSO

**rpydta** *EXCSQLSTT* (on page 381)

**cmddta** *BNDSQLSTT* (on page 136)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSET* (on page 377)  
*PRPSQLSTT* (on page 636)

**Semantic** *BNDSQLSTT* (on page 136)  
*BNDSTTASM* (on page 141)  
*DHENC* (on page 280)  
*EDTASECOVR* (on page 323)  
*ENDBND* (on page 336)  
*EUSRIDDTA* (on page 355)  
*EUSRPWDDTA* (on page 361)  
*EXCSQLIMM* (on page 371)

25. The FD:OCA description for the data value of this object is described in the DRDA Reference.

*EXCSQLSET* (on page 377)  
*PRPSQLSTT* (on page 636)  
*RTNSETSTT* (on page 794)

## NAME

SQLSTTNBR — SQL Statement Number

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD**Codepoint** X'2117'**Length** \***Class** CLASS**Sprcls** BIN - Binary Integer Number

SQL Statement Number (SQLSTTNBR) specifies the source application statement number of an SQL statement.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	8	
class	X'2117'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	32
	MINVAL	1
	REQUIRED	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

## SEE ALSO

**insvar** *BNDSQLSTT* (on page 136)**Semantic** *ENDBND* (on page 336)

**NAME**

SQLSTTVRB — SQL Statement Variable Descriptions

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2419'  
**Length** \*  
**Class** CLASS  
**Sprcls** FDOTA - FD:OCA Data

SQL Statement Variable Descriptions (SQLSTTVRB) is a byte string that specifies information about the type of application variables that appear in an SQL statement.<sup>26</sup> A description must be included for each variable in the SQL statement. The descriptions of each variable must be in the same order as they occur in the SQL statement. If a variable occurs more than once in the SQL statement, its description will occur more than once in the SQL statement.

With certain SECMECs, the DSS carrier containing the SQLSTTVRB object must be encrypted. (See *EDTASECOVR* (on page 323) for details.)

<b>clsvar</b>	NIL
<b>insvar</b>	CLASS INSTANCE VARIABLES
<b>length</b>	*
<b>class</b>	X'2419'
<b>value</b>	INSTANCE_OF BYTSTRDR - Byte String REQUIRED
<b>clscmd</b>	NIL
<b>inscmd</b>	NIL

**SEE ALSO**

**cmddta** *BNDSQLSTT* (on page 136)  
**Semantic** *BNDSQLSTT* (on page 136)  
*EDTASECOVR* (on page 323)  
*ENDBND* (on page 336)

26. The FD:OCA description for the data value of this object is described in the DRDA Reference.

**NAME**

SRVCLSNM — Server Class Name

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1147'

**Length** \*

**Class** CLASS

**Sprcls** NAME - Name

Server Class Name (SRVCLSNM) specifies the name of a class of DDM servers.

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>.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'1147'	
value	INSTANCE_OF	CHRSTRDR - Character String
	MAXLEN	255
	MINLVL	Number
	REQUIRED	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *EXCSAT* (on page 363)  
*EXCSATRD* (on page 369)

**Semantic** *PRDDTA* (on page 630)

**NAME**

SRVDGN — Server Diagnostic Information

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1153'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Server Diagnostic Information (SRVDGN) String specifies diagnostic information on reply messages that the responding server defines. This information can be logged or otherwise used to support problem determination. The contents of this parameter are unarchitected. Up to 32,767 bytes can be sent, but only a server-determined minimum amount of information should be returned.

SRVDGN should not be used in place of product-defined extensions to the architecture.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'1153'	
<b>value</b>	INSTANCE_OF	BYTSTRDR - Byte String
	MINLVL	32,767
	REQUIRED	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

- insvar** ABNUOWRM (on page 39)
- ACCRDBRM (on page 48)
- AGNPRMRM (on page 65)
- BGNBNDRM (on page 117)
- CMDATHRM (on page 171)
- CMDCHKRM (on page 173)
- CMDCMPRM (on page 175)
- CMDNSPRM (on page 176)
- CMDVLTRM (on page 181)
- CMMRQSRM (on page 182)
- DSCINVRM (on page 298)
- DTAMCHRM (on page 320)
- ENDQRYRM (on page 342)
- ENDUOWRM (on page 343)
- INTTKNRM (on page 448)
- MGRDEPRM (on page 505)
- MGRLVLRM (on page 512)
- OBINSPRM (on page 542)
- OPNQFLRM (on page 554)
- OPNQRYRM (on page 566)
- PKGBNARM (on page 585)

*PKGBPARM* (on page 586)  
*PRCCNVRM* (on page 625)  
*PRMNSPRM* (on page 633)  
*QRYNOPRM* (on page 690)  
*QRYPOPRM* (on page 691)  
*RDBACCRM* (on page 724)  
*RDBAFLRM* (on page 725)  
*RDBATHRM* (on page 727)  
*RDBNACRM* (on page 734)  
*RDBNFNRM* (on page 739)  
*RDBUPDRM* (on page 751)  
*RPYMSG* (on page 770)  
*RSCLMTRM* (on page 778)  
*RSLSETRM* (on page 785)  
*SECCHKRM* (on page 809)  
*SQLERRRM* (on page 864)  
*SYNERRCD* (on page 985)  
*SYNTAXRM* (on page 989)  
*TRGNSPRM* (on page 1022)  
*VALNSPRM* (on page 1057)

**Semantic**

*SVCERRNO* (on page 910)  
*USRSECOVR* (on page 1050)



**NAME**

SRVLCNT — Server List Count

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'244C'

**Length** \*

**Class** CLASS

**Sprcls** SCALAR - Scalar Object

Server List Count (SRVLCNT) is a count of the number of entries in the server list (SRVLST).

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	6	
<b>class</b>	X'244C'	
<b>value</b>	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	16
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *SRVLST* (on page 878)

NAME

SRVLSRV — Server List Entries

DESCRIPTION (Semantic)

**Dictionary** QDDRDBD

**Codepoint** X'244D'

**Length** \*

**Class** CLASS

**Sprcls** ORDCOL - Ordered Collection

Server List Entries (SRVLSRV) contains information for one or more target servers in the server list SRVLST. Each server list entry associates a priority with a target server as indicated by its network address. When used for workload balancing, the priority is a weighted value for the target server. When used instead for RDB failover, the priority value indicates the use of an alternate target server. The choice of TCP/IP or SNA address should be consistent with the communication layer that is being used for the conversation; that is, if the conversation is taking place over an SNA conversation, then SRVLSRV should contain an SNA address.

For a TCP/IP connection, a server list entry consists of a SRVPRTY and either an IPADDR or a TCPPOPTHOST. For an SNA connection, a server list entry consists of a SRVPRTY and an SNAADDR.

SRVLSRV consists of a list of (SRVPRTY, IPADDR) or (SRVPRTY, TCPPOPTHOST), or (SRVPRTY, SNAADDR) pairs.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'244D'	
ipaddr	INSTANCE_OF MTLEXC MTLEXC  REPEATABLE	IPADDR - TCP/IP Address X'11E9' - SNAADDR - SNA Address X'1912' - TCPPOPTHOST - TCP/IP Port Number and Domain-qualified Host Name
snaaddr	INSTANCE_OF MTLEXC MTLEXC  REPEATABLE	SNAADDR - SNA Address X'11E8' - IPADDR - TCP/IP Address X'1912' - TCPPOPTHOST - TCP/IP Port Number and Domain-qualified Host Name
srvprty	INSTANCE_OF REQUIRED NOTE REPEATABLE	SRVPRTY - Server Priority  Required with SNAADDR or IPADDR.
tcpporthost	INSTANCE_OF  MTLEXC	TCPPOPTHOST - TCP/IP Port Number and Domain-qualified Host Name X'11E8' - IPADDR - TCP/IP Address

	MTLEXC	X'11E9' - SNAADDR - SNA Address
	REPEATABLE	
	MINLVL	7
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar**      *SRVLST* (on page 878)

**NAME**

SRVLST — Server List

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'244E'**Length** \***Class** CLASS**Sprcls** ORDCOL - Ordered Collection

Server List (SRVLST) identifies the current set of servers supporting access to the same RDB. The source server uses this information for workload balancing at the site of the RDB or for RDB failover support.

**Background**

Multi-homed IP hosts and parallel host configurations have multiple addresses that can be used by a source server to connect to and access an RDB. This is motivated by needs for higher availability, capacity, and load balancing.

Another possibility is for an RDB to be replicated at one or more alternate target servers for failover support. In this scenario, a target server can return a SRVLST containing connectivity information for each alternate target server.

**Initial Connection Logic**

The source server used RDBNAM to locate an address that is either stored in the local directory or obtained from DCE or obtained from a Domain Name Server (DNS). This address provides an initial connection to a target server at the site of the RDB identified by RDBNAM. SRVLST flows on the ACCRDBRM message and contains a list of network addresses and workload priority information for the target servers associated with the RDB. The source server can use this information to either immediately connect to another target server or cache the information for use on future connections. In the case where the SRVLST is used for RDB failover support, the workload priority which is associated with an alternate target server indicates that the RDB is inactive at this server until its failover to this server has taken place.

**Caching SRVLST at Source Server**

A source server should cache SRVLSTs. Each SRVLST in the cache should be associated with the RDBNAM from the ACCRDB command.

Note that every new connect requires an ACCRDB which returns a current SRVLST.

**Subsequent Connects**

The source server should distribute connects across the server addresses in the following manner.

Let  $P$  be the value of SRVPRTY (server priority) for a server  $s$ . Let  $T$  be the sum of  $P$  for all servers in the SRVLST. The source server would distribute connections among the target servers so that the percentage of connections allocated to server  $s$  would be described by the ratio  $P/T$ .

For example, if the RDB has three target servers, *ServerA*, *ServerB*, and *ServerC* with priorities 2, 3, and 1 respectively, then  $T$  would equal 6 and the ratios  $P/T$  are 33% for *ServerA*, 50% for *ServerB*, and 17% for *ServerC*. The source server would allocate connections among the three servers

using these ratios.

When used in an environment for supporting RDB failover, the SRVPRTY for each alternate target server in the SRVLST indicates that the RDB only becomes active at the associated server once RDB failover from the original target server to this server is complete. DDM does not architect how RDB failover takes place from one target server to another. A connection to the RDB at an alternate target server will only succeed if it has become the new target server for this RDB following its failover. Therefore, the source server should only attempt to connect to the RDB at an alternate target server when it is unable to connect to the RDB at the current target server.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'244E'	
srvlcnc	INSTANCE_OF REQUIRED	SRVLCNT - Server List Count
srvlsv	INSTANCE_OF REQUIRED	SRVLSRV - Server List Entries
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

#### SEE ALSO

<b>insvar</b>	<i>ACCRDBRM</i> (on page 48)
<b>Semantic</b>	<i>SRVLCNT</i> (on page 875) <i>SRVLSRV</i> (on page 876) <i>SRVPRTY</i> (on page 883)

**NAME**

SRVNAM — Server Name

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD  
**Codepoint** X'116D'  
**Length** \*  
**Class** CLASS  
**Sprcls** MGRNAM - Manager Name

Server Name (SRVNAM) is the name of the DDM server.

No semantic meaning is assigned to server names in DDM, but it is recommended that the server's physical or logical location identifier be used as a server name.

Server names are transmitted for problem determination purposes.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
<b>length</b>	*
<b>class</b>	X'116D'
<b>value</b>	INSTANCE_OF NAMDR - Name Data REQUIRED
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO**

**insvar** EXCSAT (on page 363)  
 EXCSATRD (on page 369)  
**vldattls** SERVER (on page 824)

**NAME**

SRVOVR — Server Layer Overview

**DESCRIPTION (Semantic)**

**Dictionary** QDDTTRD

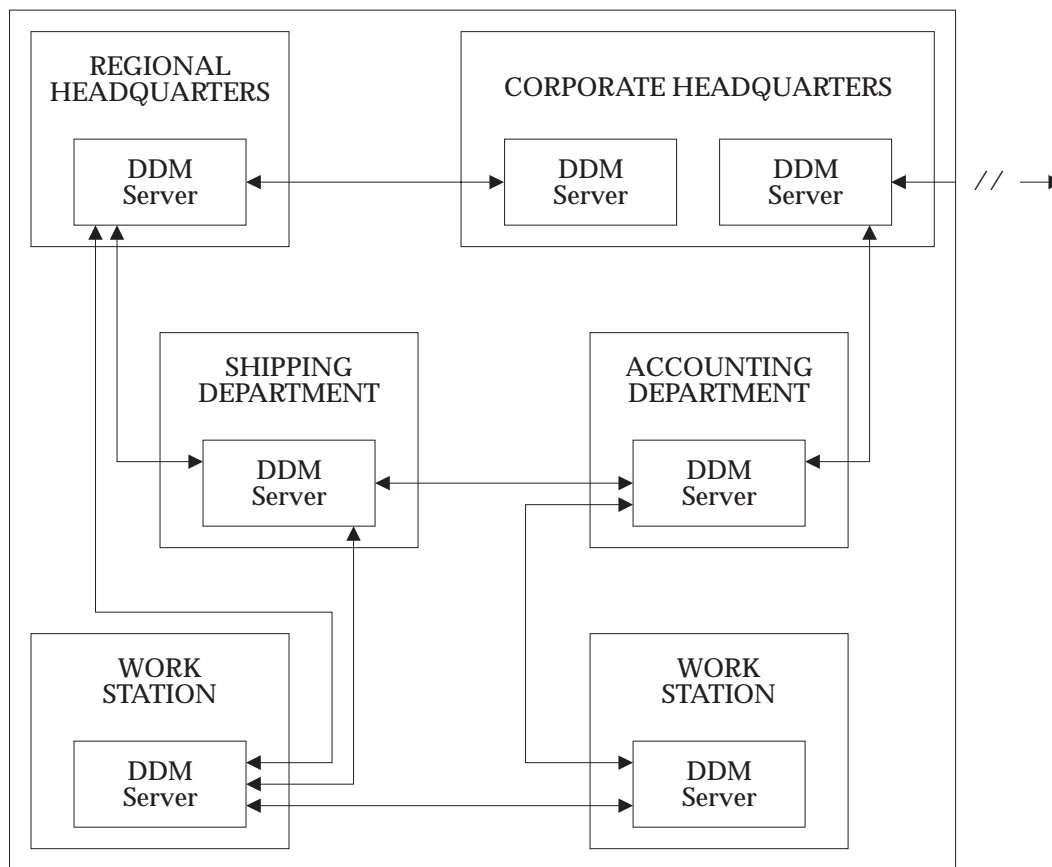
**Length** \*

**Class** HELP

Server Layer Overview (SRVOVR) describes the Distributed System and Server Layers of the DDM architecture which are illustrated in Figure 3-78. A DDM distributed system is independent of the topology (and architecture) of the underlying communications network. In fact, multiple DDM distributed systems can reside in the same communications network with servers on the same or different systems. They communicate with each other through their servers.

DDM servers can be specialized to handle specific types of resources. For example, one server can handle only files while another server handles only relational databases, and a third system handles both files and relational databases. Multiple servers can reside in the same host system.

DDM Distributed System



**Figure 3-78** DDM Distributed Systems and Servers

**SEE ALSO**

**Semantic**

*DDM* (on page 260)



**NAME**

SRVPRTY — Server Priority

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'244F'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

Server Priority (SRVPRTY) is a scalar binary number used by the source server to perform workload balancing across multiple addresses at the target server. See *SRVLST* (on page 878) for a description of workload balancing.

In an environment where the RDB is replicated at one or more alternate target server locations, the RDB is not active at an alternate target server until RDB failover from the original target server is complete. DDM does not architect how RDB failover takes place from one target server to another. In this scenario, rather than being used for load balancing, the SRVPRTY for each alternate target server is used to indicate that the RDB at the alternate target server only becomes active once RDB failover from the failed target server to this server has taken place.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	6	
<b>class</b>	X'244F'	
<b>value</b>	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	16
	NOTE	The value of 0 indicates that the RDB is not active at an alternate target server until RDB failover to this target server is complete.
	MINVAL	0
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *SRVLSRV* (on page 876)  
**Semantic** *SRVLSRV* (on page 876)  
*SRVLST* (on page 878)

**NAME**

SRVRLSLV — Server Product Release Level

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'115A'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Server Product Release Level (SRVRLSLV) String specifies the product release level of a DDM server.

The contents of this parameter are unarchitected. Up to 255 bytes can be sent.

SRVRLSLV should not be used in place of product-defined extensions to carry information not related to the product's release level.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'115A'	
value	INSTANCE_OF MAXLEN REQUIRED	BYTSTRDR - Byte String 255
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *EXCSAT* (on page 363)  
*EXCSATRD* (on page 369)

**NAME**

STGLMT — Storage Limit Reached

**DESCRIPTION (Semantic)****Dictionary** QDDBASD**Codepoint** X'1409'**Length** \***Class** codpnt

Storage Limit Reached (STGLMT) specifies that the target system reached its memory storage space limits.

**SEE ALSO****insvar** *RSCTYP* (on page 782)**Semantic** *RSCTYP* (on page 782)

## NAME

STRDELAP — String Delimiter Apostrophe

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD

**Codepoint** X'2426'

**Length** \*

**Class** codpnt

String Delimiter Apostrophe (STRDELAP) specifies that the string delimiter in the SQL statements is an apostrophe (with the Graphic Character Global Identifier (GCGID) SP05) character and that the delimiter for SQL identifiers is the double quote (with GCGID SP04) character. For more information on GCGID, see *Character Data Representation Architecture Registry* (SC09-1391, IBM).

## SEE ALSO

**insvar** *STTSTRDEL* (on page 899)

**Semantic** *STTSTRDEL* (on page 899)

**NAME**

STRDELDQ — String Delimiter Double Quote

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2427'**Length** \***Class** codpnt

String Delimiter Double Quote (STRDELDQ) specifies that the string delimiter in the SQL statements is a double quote (with the Graphic Character Global Identifier (GCGID) SP04) character and that the delimiter for SQL identifiers is an apostrophe (with the GCGID SP05) character. For more information on GCGID, see *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO****insvar** *STTSTRDEL* (on page 899)**Semantic** *STTSTRDEL* (on page 899)

**NAME**

STRING — String

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'0038'

**Length** \*

**Class** CLASS

**Sprcls** MAGNITUDE - Linearly Comparable Scalar

String (STRING) defines an ordered and contiguous collection of data values of the same class.

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>
length	*
class	X'0038'
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO**

- Semantic**
- ARMCORR* (on page 100)
  - ARRAY* (on page 101)
  - BNDCHKEXS* (on page 125)
  - BNDCRTCTL* (on page 127)
  - BNDEXPOPT* (on page 129)
  - BNDOPTNM* (on page 134)
  - BNDOPTVL* (on page 135)
  - BNDSTTASM* (on page 141)
  - CCSIDDBC* (on page 148)
  - CCSIDMBC* (on page 149)
  - CCSIDSBC* (on page 154)
  - CNNTKN* (on page 220)
  - CRRTKN* (on page 246)
  - DEPERRCD* (on page 270)
  - DIAGLVL* (on page 284)
  - DSCERRCD* (on page 296)
  - ELMCLS* (on page 330)
  - ENCALG* (on page 331)
  - ENCKEYLEN* (on page 332)
  - ENUCLS* (on page 345)
  - FDODSC* (on page 406)
  - INHERITANCE* (on page 437)
  - INSTANCE\_OF* (on page 444)
  - KERSECPPL* (on page 470)
  - LOGNAME* (on page 484)
  - LOGTSTMP* (on page 485)
  - NAME* (on page 527)
  - NOTE* (on page 533)
  - OUTOVROPT* (on page 576)

*PKGATHOPT* (on page 580)  
*PKGDFTCST* (on page 590)  
*PKGISOLVL* (on page 592)  
*PKGRPLOPT* (on page 604)  
*PKGSNLST* (on page 607)  
*PLGINID* (on page 611)  
*PLGINNM* (on page 614)  
*PLGINPPL* (on page 619)  
*PRCCNVCD* (on page 621)  
*PRDDTA* (on page 630)  
*PRDID* (on page 631)  
*QRYBLKCTL* (on page 672)  
*QRYPRCTYP* (on page 692)  
*RDBACCCL* (on page 723)  
*RDBRLSOPT* (on page 750)  
*RSCNAM* (on page 781)  
*RSCTYP* (on page 782)  
*RSNCOD* (on page 786)  
*RTNEXTDTA* (on page 792)  
*RTNSETSTT* (on page 794)  
*SECCHKCD* (on page 806)  
*SECMEC* (on page 811)  
*SECTKNOVR* (on page 822)  
*SPRCLS* (on page 832)  
*SRVDGN* (on page 873)  
*SRVRLSLV* (on page 884)  
*STTDATFMT* (on page 894)  
*STTDECDEL* (on page 897)  
*STTSTRDEL* (on page 899)  
*STTTIMFMT* (on page 900)  
*SVCERRNO* (on page 910)  
*SYNERRCD* (on page 985)  
*TEXT* (on page 1018)  
*TYPSQLDA* (on page 1034)

**NAME**

STRLYR — Structural Layers of the DDM Architecture

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

Structural Layers of DDM Architecture (STRLYR) describe how the DDM architecture uses the concepts of object-oriented programming. See *OOPOVR* (on page 547) or an overview of object-oriented programming.

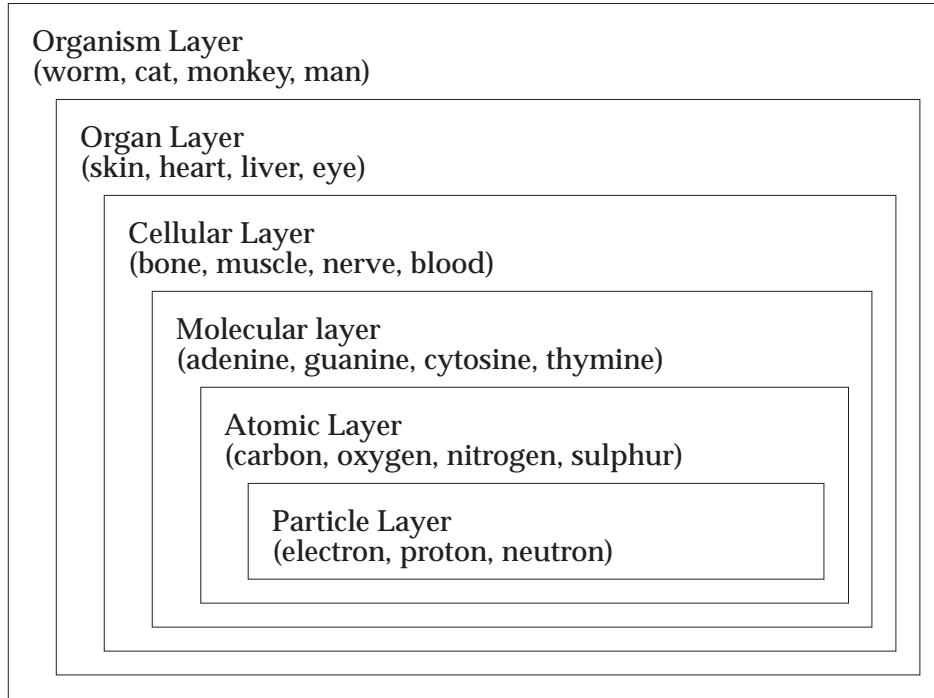
The DDM architecture seeks to define a highly uniform and concise method for processing information by objects interacting through messages. Unlike object-oriented programming languages, however, like Smalltalk where all objects exist within a uniform address space called a virtual image, the DDM architecture must support the existence and use of resources distributed throughout a network of systems. Examples of these resources are files, directories, databases, and their supporting structure. These resources exist in individual systems or subsystems called servers. Some servers support all of the classes of resources included in the DDM architecture while other servers support only a subset of those classes.

DDM uniformly models the resources, the servers, and the distributed system that the interactions of the servers define as objects that are instances of classes. Further, DDM models the attributes and data of the resources as well as the components of the messages sent between resources as objects. The DDM architecture defines a consistent framework for the existence and use of these objects.

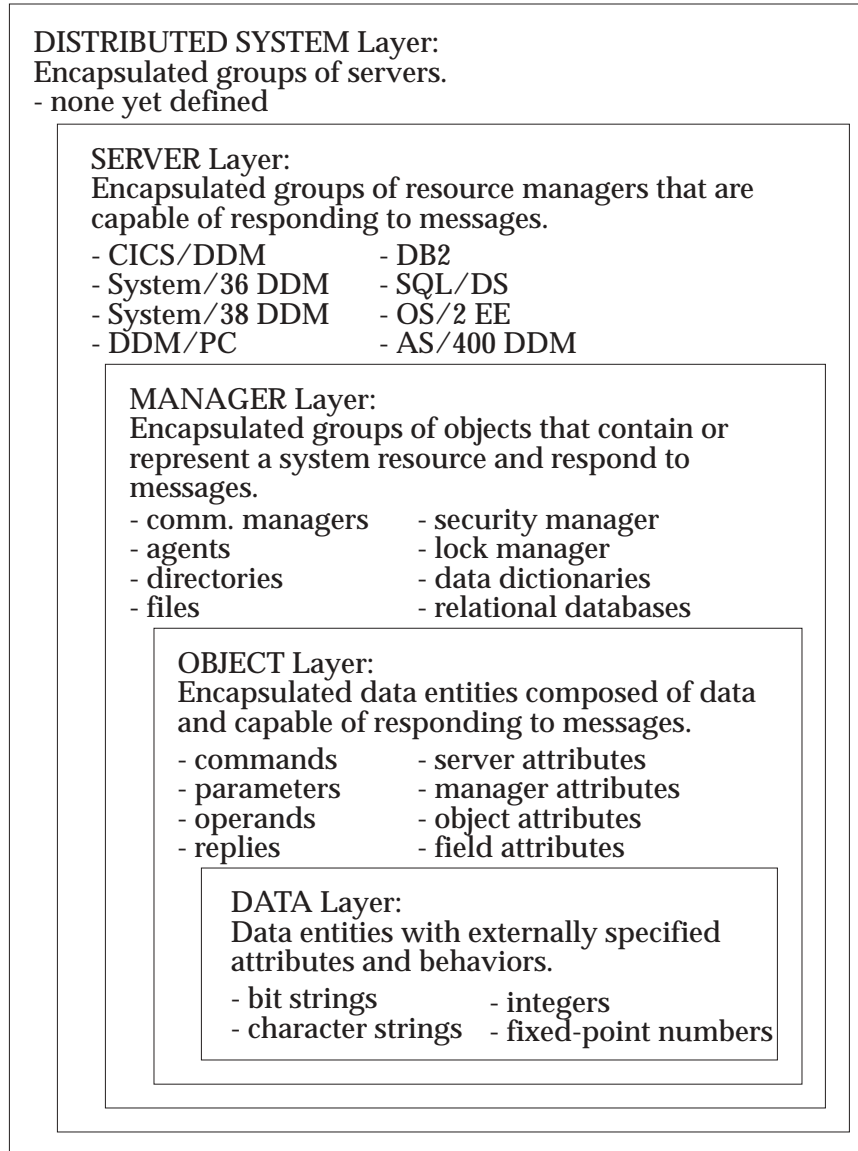
Nature and its layered composition are illustrated in Figure 3-79 (on page 891). The DDM architecture, like nature, consists of layered objects, as illustrated in Figure 3-80 (on page 892). All objects in the layers of the DDM architecture are formally modeled as objects defined by classes. While a single class-oriented method describes these objects, the classes defined for each layer form separate, independent hierarchies. For example, manager-layer objects do not inherit characteristics from either object-layer or server-layer classes.

The remainder of this term examines the layers of the DDM architecture.





**Figure 3-79** Structural Layers in Nature



**Figure 3-80** The Structural Layers of DDM Architecture

**SEE ALSO**

**Semantic**

*CONCEPTS* (on page 243)

**NAME**

STTASMEUI — Statement Assumptions Executable Unique Section Input

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2437'

**Length** \*

**Class** codpnt

Statement Assumptions Executable Unique Section Input (STTASMEUI) specifies that the source server program preparation process (precompiler) was not able to classify the SQL statement; therefore, the following assumptions are made about the SQL statement:

- The statement can be executed with an EXCSQLSTT command.
- This SQL statement was assigned a unique section number and no other SQL statement shares that section.
- All variables contained in the SQL statement are assumed to be input variables to the relational database.

**SEE ALSO**

**insvar** *BNDSTTASM* (on page 141)

## NAME

STTDATFMT — Statement Date Format

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD**Codepoint** X'2122'**Length** \***Class** CLASS**Sprcls** STRING - String

Statement Date Format (STTDATFMT) String specifies the date format used in the SQL statements.

The ISODATFMT and JISDATFMT specify a common date format. They are kept separate for reporting purposes and to keep the encoding consistent with the statement time format (STTTIMFMT) which is different.

clsvar	NIL	
insvar	CLASS INSTANCE VARIABLES	
length	6	
class	X'2122'	
value	INSTANCE_OF	CODPNTDR - Codepoint Data Representation
	ENUVAL	X'2429' - ISODATFMT - ISO Date Format
	ENUVAL	X'242A' - USADATFMT - USA Date Format
	ENUVAL	X'242B' - EURDATFMT - European Date Format
	ENUVAL	X'242C' - JISDATFMT - Japanese Industrial Standard Date Format
	ENUVAL	X'2400' - DFTDATFMT - Default Date Format
	MINLVL	7
	ENUVAL	X'2448' - LOCDATFMT - Local Date Format
	MINLVL	7
	ENUVAL	X'2402' - DMYBLKDATFMT - DMY with Blank Separator Date Format
	MINLVL	7
	ENUVAL	X'2403' - DMYCMADATFMT - DMY with Comma Separator Date Format
	MINLVL	7
	ENUVAL	X'2404' - DMYHPNDATFMT - DMY with Hyphen Separator Date Format
	MINLVL	7
	ENUVAL	X'2406' - DMYPRDDATFMT - DMY with Period Separator Date Format
	MINLVL	7
	ENUVAL	X'2409' - DMYSLHDATFMT - DMY with Slash Separator Date Format
	MINLVL	7
	ENUVAL	X'242D' - JULBLKDATFMT - Julian with

		Blank Separator Date Format
MINLVL	7	
ENUVAL	X'243F'	- JULCMADATFMT - Julian with Comma Separator Date Format
MINLVL	7	
ENUVAL	X'2440'	- JULHPNDATFMT - Julian with Hyphen Separator Date Format
MINLVL	7	
ENUVAL	X'2446'	- JULPRDDATFMT - Julian with Period Separator Date Format
MINLVL	7	
ENUVAL	X'2447'	- JULSLHDATFMT - Julian with Slash Separator Date Format
MINLVL	7	
ENUVAL	X'244A'	- MDYBLKDATFMT - MDY with Blank Separator Date Format
MINLVL	7	
ENUVAL	X'244B'	- MDYCMADATFMT - MDY with Comma Separator Date Format
MINLVL	7	
ENUVAL	X'2451'	- MDYHPNDATFMT - MDY with Hyphen Separator Date Format
MINLVL	7	
ENUVAL	X'2452'	- MDYPRDDATFMT - MDY with Period Separator Date Format
MINLVL	7	
ENUVAL	X'2453'	- MDYSLHDATFMT - MDY with Slash Separator Date Format
MINLVL	7	
ENUVAL	X'2454'	- YMDBLKDATFMT - YMD with Blank Separator Date Format
MINLVL	7	
ENUVAL	X'2455'	- YMDCMADATFMT - YMD with Comma Separator Date Format
MINLVL	7	
ENUVAL	X'2456'	- YMDHPNDATFMT - YMD with Hyphen Separator Date Format
MINLVL	7	
ENUVAL	X'2457'	- YMDPRDDATFMT - YMD with Period Separator Date Format
MINLVL	7	
ENUVAL	X'2458'	- YMDSLHDATFMT - YMD with Slash Separator Date Format
MINLVL	7	
DFTVAL	X'2429'	- ISODATFMT - ISO Date Format
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

## SEE ALSO

**insvar**

*BGNBND* (on page 110)  
*CODPNTDR* (on page 235)  
*DFTDATFMT* (on page 272)  
*DMYBLKDATFMT* (on page 288)  
*DMYCMADATFMT* (on page 289)  
*DMYHPNDATFMT* (on page 290)  
*DMYPRDDATFMT* (on page 291)  
*DMYSLHDATFMT* (on page 292)  
*EURDATFMT* (on page 353)  
*ISODATFMT* (on page 451)  
*JISDATFMT* (on page 458)  
*JULBLKDATFMT* (on page 460)  
*JULCMADATFMT* (on page 461)  
*JULHPNDATFMT* (on page 462)  
*JULPRDDATFMT* (on page 463)  
*JULSLHDATFMT* (on page 464)  
*LOCDATFMT* (on page 482)  
*MDYBLKDATFMT* (on page 500)  
*MDYCMADATFMT* (on page 501)  
*MDYHPNDATFMT* (on page 502)  
*MDYPRDDATFMT* (on page 503)  
*MDYSLHDATFMT* (on page 504)  
*STTTIMFMT* (on page 900)  
*USADATFMT* (on page 1041)  
*YMDBLKDATFMT* (on page 1113)  
*YMDCMADATFMT* (on page 1114)  
*YMDHPNDATFMT* (on page 1115)  
*YMDPRDDATFMT* (on page 1116)  
*YMDSLHDATFMT* (on page 1117)

**Semantic**

*CLASS* (on page 158)  
*ENDBND* (on page 336)  
*ISODATFMT* (on page 451)  
*JISDATFMT* (on page 458)  
*QDDRDBD* (on page 657)  
*REBIND* (on page 753)  
*STRING* (on page 888)  
*STTTIMFMT* (on page 900)

**NAME**

STTDECDEL — Statement Decimal Delimiter

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2121'  
**Length** \*  
**Class** CLASS  
**Sprcls** STRING - String

Statement Decimal Delimiter (STTDECDEL) String specifies the character used as the decimal delimiter in SQL statements.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	6	
class	X'2121'	
value	INSTANCE_OF	CODPNTDR - Codepoint Data Representation
	ENUVAL	X'243C' - DECDELPRD - Decimal Delimiter Is Period
	ENUVAL	X'243D' - DECDELCMA - Decimal Delimiter Is Comma
	ENUVAL	X'241E' - DFTPKG - Package Default
	NOTE	This value is not valid when the parameter is used on BGNBND command.
	DFTVAL	''
	NOTE	Means that the target system defined default decimal delimiter is used.
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** ACCRDB (on page 42)  
 BGNBND (on page 110)  
**Semantic** DFTPKG (on page 273)  
 REBIND (on page 753)

**NAME**

STTSCCCLS — Statement Successfully Classified

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2436'**Length** \***Class** codpnt

Statement Successfully Classified (STTSCCCLS) specifies that the source server program preparation process (precompiler) properly classified the SQL statement, and therefore, assumptions about the SQL statement are not necessary.

**SEE ALSO****insvar** *BNDSTTASM* (on page 141)



**NAME**

STTSTRDEL — Statement String Delimiter

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD  
**Codepoint** X'2120'  
**Length** \*  
**Class** CLASS  
**Sprcls** STRING - String

Statement String Delimiter (STTSTRDEL) specifies which separate characters delimit character strings and delimited SQL identifiers in SQL statements.

If the value of this parameter is STRDELAP, then the string delimiter in SQL statements is an apostrophe character, and the delimiter for delimited SQL delimiters is a double quote character.

If the value of this parameter is STRDELQ, then the string delimiter in SQL statements is a quote character, and the delimiter for SQL statements is an apostrophe character.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	6	
class	X'2120'	
value	INSTANCE_OF	CODPNTDR - Codepoint Data Representation
	ENUVAL	X'2426' - STRDELAP - String Delimiter Apostrophe
	ENUVAL	X'2427' - STRDELQ - String Delimiter Double Quote
	ENUVAL	X'241E' - DFTPKG - Package Default
	NOTE	This value is not valid when the parameter is used on BGNBND command.
	DFTVAL	''
	NOTE	Means that the target RDB default characters are used to delimit character strings and delimited SQL identifiers in SQL statements.
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** ACCRDB (on page 42)  
 BGNBND (on page 110)  
**Semantic** DFTPKG (on page 273)  
 ENDBND (on page 336)  
 REBIND (on page 753)

## NAME

STTTIMFMT — Statement Time Format

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD**Codepoint** X'2123'**Length** \***Class** CLASS**Sprcls** STRING - String

The Statement Time Format (STTTIMFMT) String specifies the time format used in the SQL statements.

The ISOTIMFMT and EURTIMFMT specify a common time format. They are kept separate for reporting purposes and to keep the encoding consistent with the statement date format (STTDATFMT) which is different.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	6	
<b>class</b>	X'2123'	
<b>value</b>	INSTANCE_OF	CODPNTDR - Codepoint Data Representation
	ENUVAL	X'242E' - ISOTIMFMT - ISO Time Format
	ENUVAL	X'242F' - USATIMFMT - USA Time Format
	ENUVAL	X'2430' - EURTIMFMT - European Time Format
	ENUVAL	X'2431' - JISTIMFMT - Japanese Industrial Standard Time Format
	ENUVAL	X'2401' - DFTTIMFMT - Default Time Format
	MINLVL	7
	ENUVAL	X'2449' - LOCTIMFMT - Local Time Format
	MINLVL	7
	ENUVAL	X'240C' - HMSBLKTIMFMT - HMS with Blank Separator Time Format
	MINLVL	7
	ENUVAL	X'240D' - HMSCLNTIMFMT - HMS with Colon Separator Time Format
	MINLVL	7
	ENUVAL	X'2416' - HMSCMATIMFMT - HMS with Comma Separator Time Format
	MINLVL	7
	ENUVAL	X'2428' - HMSPRDTIMFMT - HMS with Period Separator Time Format
	MINLVL	7
	DFTVAL	X'242E' - ISOTIMFMT - ISO Time Format
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

<b>insvar</b>	<i>BGNBND</i> (on page 110) <i>CODPNTDR</i> (on page 235) <i>DFTTIMFMT</i> (on page 275) <i>EURTIMFMT</i> (on page 354) <i>HMSBLKTIMFMT</i> (on page 431) <i>HMSCLNTIMFMT</i> (on page 432) <i>HMSCMATIMFMT</i> (on page 433) <i>HMSPRDTIMFMT</i> (on page 434) <i>ISOTIMFMT</i> (on page 457) <i>JISTIMFMT</i> (on page 459) <i>LOCTIMFMT</i> (on page 483) <i>USATIMFMT</i> (on page 1042)
<b>Semantic</b>	<i>CLASS</i> (on page 158) <i>EURTIMFMT</i> (on page 354) <i>ISOTIMFMT</i> (on page 457) <i>QDDRDBD</i> (on page 657) <i>REBIND</i> (on page 753) <i>STRING</i> (on page 888) <i>STTDATFMT</i> (on page 894)

## NAME

SUBSETS — Architecture Subsets

## DESCRIPTION (Semantic)

**Dictionary** QDDTTRD**Length** \***Class** HELP

Architecture Subsets (SUBSETS) describe the goal and rules for subsetting the DDM architecture.

**Goal for Subsetting the DDM Architecture**

The goal of DDM is to maximize data connectivity among the products that implement the architecture. However, it is not reasonable:

- To require all products to support all of the architected file classes, access methods, utilities, and their associated commands
- To require all products to support a relational database, SQL application manager, and their associated commands
- To allow products to select among the file classes, access methods, relational databases, commands, and capabilities of DDM for only those that match exactly the capabilities of their local data management facilities
- To restrict products to only the architected constructs of DDM

Therefore, the DDM architecture includes rules that guide product programmers:

- In selecting subsets of the DDM architecture
- In developing product-unique extensions to the DDM architecture

Natural boundaries within the overall architecture are necessary to subsetting. These boundaries have been determined, and subsets have been designed into DDM. Each subset consists of a number of classes of objects and the commands to which they respond.

DDM does not strive for one-hundred percent local/remote transparency for all local data management system. Even with a subsetting policy, an exact match may not exist between DDM subsets and local data management. In order to achieve local/remote transparency and functional match with other products, a product may need:

- To emulate DDM commands on a target by using a sequence of target data management requests
- To emulate source data management functions on a target system by sending a chain of DDM commands

**Subsetting Rules**

The following subsetting rules have been adopted for the DDM architecture:

1. Select the class of *server* to be implemented. See *SERVER* (on page 824). Servers are the logical units of control and access for a set of data resources. A server is classified by the type of system or operating system that provides its local processing environment.

Servers are not, however, synonymous with a system product or its hardware. A product and its successor could implement the same server even though they might run on different hardware. Alternately, multiple copies of an application server could run on the

same hardware.

- If a server acts only as a source of requests (as a source server) and does not provide remote data management services for remote requesters (as a target server), then no additional DDM support is required. The translation of local data management interfaces to the canonical language of the DDM architecture is not part of the DDM architecture.

- All target servers must support the following DDM manager classes:<sup>27</sup>

Supervisor	See <i>SUPERVISOR</i> (on page 908).
Security Manager	The level of security DDM defines is required of all target servers. See <i>SECMGR</i> (on page 814).
Dictionary	The tables of descriptors for codepoint identified descriptors of transmitted DDM terms and product extension terms. See <i>DICTIONARY</i> (on page 286). The required DDM dictionaries (QDDPRMD and QDDBASD) must be supported.

Support of the DDM dictionary, QDDRDBD, is optional. It must be supported if the server provides relational database functions.

The DDM architecture is documented in terms of dictionaries of formal class description objects. Products are not required to implement either dictionaries or class descriptors as documented in the architecture. Products are only required to implement the semantic equivalents of dictionaries and classes.

2. Select the communications managers to be supported. Each DDM communications manager is designed to support a single protocol for a single communications facility. There are several communications managers defined:

CMNAPPC	LU 6.2 Conversational Communications Manager (see <i>CMNAPPC</i> (on page 184))
CMNSYNCPT	SNA LU 6.2 Sync Point Conversational Communications Manager (see <i>CMNSYNCPT</i> (on page 202))
CMNTCPIP	TCP/IP Communication Manager (see <i>CMNTCPIP</i> (on page 214))

If the CMNSYNCPT is selected, then the SYNCPTMGR must also be selected.

3. Select the DDM agent class to be supported. DDM agents represent the requester on both the source system and the target system. The source agent is bound to a target agent to actually request services from the target and return replies to the source. See *AGENT* (on page 61) for a description of agent responsibilities. Only one class of agents is defined.
4. Select the architected database class being supported. The only database class defined in DDM is RDB for relational databases.

If a database class is supported then it is required to support a database access class.

---

<sup>27</sup>. Prior to DDM Level 3, target servers were also required to support the lock manager.

A target server is required to support the database class.

5. Select the database access class being supported. The only database access class defined in DDM is SQLAM.
6. Select the system command class to be supported. The only DDM manager that handles system commands is the system command manager (SYSCMDMGR).
7. Select one or more managers to support resource recovery if needed. The following managers are defined:

SQLAM            SQL Application Manager (see *SQLAM* (on page 847))

The SQLAM uses the RDBCMM and RDBRLLBCK commands to perform one-phase commit processing. If the SQLAM is selected, only one target server may have its recoverable resources protected. Further, the CMNSYNCPT cannot be used with the SQLAM.

SYNCPTMGR    Sync Point Manager (see *SYNCPTMGR* (on page 939))

The SYNCPTMGR uses the two-phase commit process. If the SYNCPTMGR is selected, then the source server, and many target servers may have their recoverable resources protected. Further, the CMNSYNCPT must be selected. The SYNCPTMGR overrides the SQLAM single-phase commit process; that is, RDBCMM and RDBRLLBCK are not allowed.

XAMGR            XA Manager (see *XAMGR* (on page 1063))

The XAMGR provides an interface to allow the application to protect its own resources. If XAMGR is selected, then the source server, and many target servers, may have their recoverable resources protected. The SYNCPTMGR and XAMGR are mutually-exclusive. An XAMGR Global Transaction overrides the SQLAM single-phase commit process; that is, RDBCMM and RDBRLLBCK are not allowed. An XAMGR Local Transaction does not override the SQLAM single-phase commit process; that is, RDBCMM and RDBRLLBCK must be used.

All managers can be selected; however, only one of them can protect a given unit of work.

In addition to the architected classes and commands of DDM, products are encouraged to design classes and commands to meet product-unique requirements for horizontal growth or function distribution. DDM includes support for such *open architecture* enhancement. To the extent possible, these extensions should be fed back into the DDM architecture development process for cross-product standardization. Existing DDM classes, commands, parameters, and messages can be used in the definition of product extensions to DDM and can be intermixed with product-defined structures as needed.

The following requirements must be met in developing product extensions:

1. All products must support connections with other products based solely on architected DDM codepoints in the dictionaries QDDPRMD, QDDBASD, and QDDRDBD. While product-pair agreements to support product extensions are permitted, they cannot be required.
2. Product extension codepoints can be sent only if both the source and target servers have the same product extension dictionary in their dictionary lists.

When selecting the manager classes to be supported, the manager dependency list of each manager must be checked to ensure that all dependent managers have been selected for support

at the specified manager level (or greater).

### Commands, Parameters, Values, and Objects

Each target server is required to reply to unrecognized and unsupported commands, parameters, parameter values, and command objects with one of the following reply messages:

CMDNSPRM	Command Not Supported (see <i>CMDNSPRM</i> (on page 176))
OBJNSPRM	Object Not Supported (see <i>OBJNSPRM</i> (on page 542))
PRMNSPRM	Parameter Not Supported (see <i>PRMNSPRM</i> (on page 633))
TRGNSPRM	Target Not Supported (see <i>TRGNSPRM</i> (on page 1022))
VALNSPRM	Parameter Value Not Supported (see <i>VALNSPRM</i> (on page 1057))

Each source server is required to detect unrecognized or unsupported reply messages or reply objects. DDM has neither an architected mechanism for the source server to inform the target server of these problems, nor is there any required recovery action for them. However, the source server assumes that the DDM conversation has failed and takes implementation-defined recovery action.

For both source and target servers, the DDM architecture prescribes the means for recognizing the codepoints of valid DDM structures. These are:

1. The basis for recognizing a command is the class of object to which it is sent (see *AGNCMDPR* (on page 64)). Every command in DDM has one parameter with the *command target* (CMDTRG) attribute. This parameter names the object to which the command is being sent.

Every target object must be an instance of a DDM class or of a product extension class. If a specified target object is not an instance of a DDM class, of a product extension class, or of a DDM class the target server supports, then the TRGNSPRM reply message must be returned.

The class description of the target object includes an INSTANCE COMMANDS list that specifies the commands supported for the target object. If the target object is itself a class, use the CLASS COMMANDS list of the target. If the command sent is not in that list, the CMDNSPRM is returned.

The CMDNSPRM cannot be returned for the REQUIRED commands of a target object.

The CMDNSPRM must be returned:

- For the unsupported OPTIONAL commands of a target object
  - For commands not found in the class description of the target
  - For product extension commands to the target object if no previous agreement between the servers to use the product extension exists
2. The basis for recognizing a parameter is the INSTANCE VARIABLES list of the command for which it is sent. This list specifies the valid parameters for the command.

The PRMNSPRM cannot be returned for a parameter found in the parameter list of the command's class description.

The PRMNSPRM must be returned:

- If the parameter is not found in the parameter list of the class description of the command

- For product extension parameters if no previous server agreement to use the product extension exists
3. The basis for recognizing a parameter's value is the INSTANCE VARIABLES list of the parameter's and command's class description. Values specified in the instance variables list of the command term take precedence over the values specified in the parameter term. The valid values of the parameter are defined in terms of one or more data fields. Each data field is defined in terms of a list of attributes of valid values for the field. For example, if a list of enumerated values (ENUVAL attribute) is specified, the field value sent must be one of the listed values.

The VALNSPRM cannot be returned:

- For parameters that the target can ignore
- For parameters for which there is an architected promotion scheme and when the target server supports the promotion value (for instance, all servers must support some form of locking or lock promotion)

The VALNSPRM must be returned:

- If any field of the value sent violates any of the attributes listed by the parameter's class description (for instance, if the value sent is 500, but the maximum value allowed is 255)
  - If the value sent exceeds the implementation capabilities of the target server (for instance, if the initial size specified for a new file exceeds the storage capabilities of the target server)
  - If the value sent requests or implies a function that the target system cannot perform (for instance, if a file is created to have variable length records, but the target system does not support the variable length record class)
  - For product extension values if no previous server agreement to use the product extension exists
4. The basis for recognizing objects sent as command data is the COMMAND OBJECTS list of the command's class description with the same request correlation identifier (see *DSS* (on page 308) and *RQSCRR* (on page 772)). The class description of each command specifies the list of object classes that can be sent as command data.

The OBJNSPRM cannot be returned for the REQUIRED data objects of the command.

The target server must return the OBJNSPRM:

- If the class of a data object is not found in the command data list
  - If the target server does not support the command for the class of object sent
  - If the target object of the command does not support the class of object sent
  - For product extension objects if no previous server agreement to use the product extension exists
5. The basis for recognizing an object as a member of a collection object is the INSTANCE VARIABLES list of the collection's class description.

The OBJNSPRM cannot be returned for the REQUIRED objects of the collection.

The target server must return the OBJNSPRM:



- If the class of a data object is not found in the collection's INSTANCE VARIABLE list
  - If the target server does not support the object in the current context of the command and command target
  - For product extension objects if no previous server agreement to use the product extension exists
6. The basis for recognizing a value of a scalar COMMAND OBJECT is defined in item 3 above for the value of parameters. The VALNSPRM can be returned as defined in item 3.
  7. The basis for recognizing objects sent in reply to a command is the REPLY OBJECTS list of the command's class description with the same request correlation identifier (see *DSS* (on page 308) and *RQSCRR* (on page 772)). The class description of each command specifies the exclusive list of object classes that can be returned as reply data.  
A product extension reply object cannot be returned if no previous server agreement to use the product extension dictionaries exists.
  8. The basis for recognizing an object of a collection object sent as a REPLY OBJECT is the INSTANCE VARIABLES list of the collection's class description. An example of such a collection is a RECAL.  
A product extension object cannot be returned in the collection if no previous server agreement to use the product extension exists.
  9. The basis for recognizing a value of a scalar REPLY OBJECT is the INSTANCE VARIABLES list of the object's class description. An example of such a collection is a RECAL.  
A product extension value cannot be returned in the collection if no previous server agreement to use the product extension exists.
  10. The basis for recognizing reply messages is the COMMAND REPLIES list of the command's class description with the same request correlation identifier (see *DSS* (on page 308) and *RQSCRR* (on page 772)). The class description of each command specifies the exclusive list of reply messages that can be returned.  
A product extension reply message cannot be returned if no previous server agreement to use the product extension dictionaries exists.

**SEE ALSO**

<b>mgrdepls</b>	<i>MANAGER</i> (on page 491)
<b>Semantic</b>	<i>CMDNSPRM</i> (on page 176) <i>CONCEPTS</i> (on page 243) <i>EXTENSIONS</i> (on page 400) <i>OPTIONAL</i> (on page 568) <i>PCOVR</i> (on page 628) <i>PRMNSPRM</i> (on page 633) <i>REQUIRED</i> (on page 761) <i>VALNSPRM</i> (on page 1057)

**NAME**

SUPERVISOR — Supervisor

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'143C'

**Length** \*

**Class** CLASS

**Sprcls** MANAGER - Resource Manager

Supervisor (SUPERVISOR) is a basic operative part of a DDM server. A supervisor manages a collection of managers in a consistent manner. A server has one, and only one, supervisor.

Supervisors provide an interface to their local system services, such as resource management, directory, dictionary, and security services. The DDM supervisor interfaces with the local system services and the other DDM managers.

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

**Manager-Level Compatibility**

Table 3-16 illustrates the function of the SUPERVISOR. It has not grown or changed through the levels of the DDM architecture. The EXCSAT command also has not changed in its basic structure. The manager-level list parameter is an open-ended list. See *EXCSAT* (on page 363).

**Table 3-16** Supervisor-Level Compatibility

<b>DDM Levels</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
SUPERVISOR	1	1	1	1	1
<i>Commands</i>					
EXCSAT	1	1	1	1	1

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'143C'	
secmgr	INSTANCE_OF	SECMGR - Security Manager
<b>clscmd</b>	NIL	
<b>inscmd</b>	INSTANCE COMMANDS	
1041	INSTANCE_OF REQUIRED	EXCSAT - Exchange Server Attributes
<b>mgrlvl</b>	1	
<b>mgrdepls</b>	NIL	
<b>vldattls</b>	VALID ATTRIBUTES	

0019	INSTANCE_OF	HELP - Help Text
115D	INSTANCE_OF	SPVNAM - Supervisor Name
0045	INSTANCE_OF	TITLE - Title

**SEE ALSO**

<b>insvar</b>	<i>DEPERRCD</i> (on page 270) <i>MGRLVL</i> (on page 506)
<b>Semantic</b>	<i>ACCRDB</i> (on page 42) <i>AGENT</i> (on page 61) <i>EXTENSIONS</i> (on page 400) <i>INHERITANCE</i> (on page 437) <i>MANAGER</i> (on page 491) <i>RDBOVR</i> (on page 740) <i>SUBSETS</i> (on page 902)

**NAME**

SVCERRNO — Security Service Error Number

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'11B4'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Security Service Error Number (SVCERRNO) contains an error code from the local security service. SRVDGN may contain additional information.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'11B4'	
value	INSTANCE_OF LENGTH	BINDR - Binary Number Field 32
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *SECCHKRM* (on page 809)

**NAME**

SVRCOD — Severity Code

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1149'

**Length** \*

**Class** CLASS

**Sprcls** BIN - Binary Integer Number

Severity Code (SVRCOD) is an indicator of the severity of a condition detected during the execution of a command.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	6	
<b>class</b>	X'1149'	
<b>value</b>	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	16
	ENUVAL	0 - INFO - Information Only Severity Code
	ENUVAL	4 - WARNING - Warning Severity Code
	ENUVAL	8 - ERROR - Error Severity Code
	ENUVAL	16 - SEVERE - Severe Error Severity Code
	ENUVAL	32 - ACCDMG - Access Damage Severity Code
	ENUVAL	64 - PRMDMG - Permanent Damage Severity Code
	ENUVAL	128 - SESDMG - Session Damage Severity Code
	REQUIRED	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

- insvar** ABNUOWRM (on page 39)
- ACCRDBRM (on page 48)
- ACCSECRD (on page 58)
- AGNPRMRM (on page 65)
- BGNBNDRM (on page 117)
- CMDATHRM (on page 171)
- CMDCHKRM (on page 173)
- CMDCMPRM (on page 175)
- CMDNSPRM (on page 176)
- CMDVLTRM (on page 181)
- CMMRQSRM (on page 182)
- DSCINVRM (on page 298)
- DTAMCHRM (on page 320)
- ENDQRYRM (on page 342)
- ENDUOWRM (on page 343)
- INTTKNRM (on page 448)
- MGRDEPRM (on page 505)
- MGRLVLRM (on page 512)

*OBJNSPRM* (on page 542)  
*OPNQFLRM* (on page 554)  
*OPNQRYRM* (on page 566)  
*PKGBNARM* (on page 585)  
*PKGBPARM* (on page 586)  
*PRCCNVRM* (on page 625)  
*PRMNSPRM* (on page 633)  
*QRYNOPRM* (on page 690)  
*QRYPOPRM* (on page 691)  
*RDBACCRM* (on page 724)  
*RDBAFLRM* (on page 725)  
*RDBATHRM* (on page 727)  
*RDBNACRM* (on page 734)  
*RDBNFNRM* (on page 739)  
*RDBUPDRM* (on page 751)  
*RPYMSG* (on page 770)  
*RSCLMTRM* (on page 778)  
*RSLSETRM* (on page 785)  
*SECCHKRM* (on page 809)  
*SQLERRRM* (on page 864)  
*SYNERRCD* (on page 985)  
*SYNTAXRM* (on page 989)  
*TRGNsprm* (on page 1022)  
*VALNSPRM* (on page 1057)

**Semantic**

*ACCSECRD* (on page 58)  
*APPTRGER* (on page 95)  
*DCESECOVR* (on page 253)  
*EDTASECOVR* (on page 323)  
*LMTBLKPRC* (on page 475)  
*PLGINOVR* (on page 615)  
*SECCHK* (on page 800)  
*SECCHKCD* (on page 806)  
*SECCHKRM* (on page 809)  
*TCPTRGER* (on page 1015)  
*USRSECOVR* (on page 1050)

**NAME**

SYNCCRD — Sync Point Control Reply

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD  
**Codepoint** X'1248'  
**Length** \*  
**Class** CLASS  
**Sprcls** COLLECTION - Collection Object

SYNCCRD conveys sync point information to the source SYNCPTMGR or XAMGR.

See *RLSCONV* (on page 765) or *XAMGROV* (on page 1066) for the use of this parameter with the XAMGR.

**Release for Reuse**

If the source SYNCPTMGR requests the connection be released for reuse at the end of the commit or rollback by specifying *rlsconv(reuse)*, the RDB should be requested to provide an indication if the connection can be reused by another application. If specified in the SYNCCTL command, the *rlsconv* instance variable must be returned in the reply with the appropriate indication.

The target RDB can generate a group of SQL SET statements to be used to re-establish the current execution environment if another transaction is executed for the application. These SQLSTT objects are returned with the SYNCCRD. When a release for reuse connection is reused, the source server must issue the same commands used to establish the connection, access the RDB, and then issue the set execution environment command with any SQL SET statements returned on the last commit or rollback executed on behalf of the application (that is, EXCSAT, ACCSEC, SECCHK, ACCRDB, EXCSQLSET). Prior to the connection being reused, the RDB 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.

**DSS Carrier: OBJDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	8	
<b>class</b>	X'1248'	
<b>prphrclst</b>	INSTANCE_OF	PRPHRCLST - List of Prepared and Heuristically Completed XIDs
	OPTIONAL	
	NOTE	Parameter only valid with the XAMGR; otherwise, should not be specified.
		For XAMGR, parameter is only valid if responding to the SYNCCTL(Return Indoubt list) request.

	MINLVL	XAMGR (Level 7)
synctype	INSTANCE_OF REQUIRED NOTE	SYNCTYPE - Type of Sync Point Operation  A value of none, X'00', can be specified when no other synctype applies, such as on an unprotected connection.  For SYNCPTMGR, this parameter is required.  For XAMGR, this parameter is not valid and should not be specified.
	MINLVL	SYNCPTMGR (Level 5)
xaretval	INSTANCE_OF REQUIRED NOTE	XARETVAL - XA Return Value  Parameter only valid with the XAMGR; otherwise, should not be specified.  For XAMGR, when replying to the SYNCCTL(commit), the values starting with XA_RB* can only be specified if the XAFLAG(TMONPHASE) was sent on the request.
	DFTVAL	X'00000000' - XA_OK
	MINLVL	XAMGR (Level 7)

**SEE ALSO**

- insvar**            *FORGET* (on page 423)
- rpydta**           *SYNCCTL* (on page 915)
- Semantic**        *FORGET* (on page 423)  
                      *SYNCCTL* (on page 915)  
                      *SYNCPTOV* (on page 944)  
                      *XAMGR* (on page 1063)  
                      *XAMGROV* (on page 1066)



**NAME**

SYNCCTL — Sync Point Control Request

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1055'

**Length** \*

**Class** CLASS

**Sprcls** COMMAND - Command

Sync Point Control Command (SYNCCTL) conveys sync point information to the target SYNCPTMGR or target XAMGR.

**Source System Processing****SYNCPTMGR Support**

The source SYNCPTMGR uses the SYNCCTL command in the following situations:

- To request log information from the target SYNCPTMGR
- To send a new unit of work identifier to start a new unit of work (begin a transaction) at the target SYNCPTMGR
- During sync point processing by the source SYNCPTMGR
- To terminate a released conversation
- To request if the target server can reuse the connection for another application

A sync point request log command must be sent before any commands are sent to the RDB.

The *uowid* parameter on the sync point new unit of work command must be sent to each target SYNCPTMGR prior to sending any DDM commands targeted to the RDB. This command indicates that the target SYNCPTMGR is a participant in the current unit of work and will participate in the sync point operation. No reply is expected on the sync point new unit of work command. If the application requires resources to be shared between a set of SYNCPTMGR protected connections, the source SYNCPTMGR can send an XID parameter on a *syncctl* (new unit or work) to identify to the target RDB which SYNCPTMGR protected connections are allowed to share resources in order to prevent deadlocks from occurring between them.

A sync point prepare command may require SYNCLOG command data to be sent to the target SYNCPTMGR. If this information has already been sent on a previous sync point operation and the information has not changed, the SYNCLOG command object is optional.

The *release* parameter on the sync point prepare to commit or request to commit command is used to request the target server to terminate this conversation when the sync point operation completes and the decision is to commit the unit of work.

If the conversation is to be terminated at the completion of the sync point operation, the *release* parameter is set to TRUE on the sync point prepare command or for the case of the resync server, the sync point request to commit command. The conversation cannot be terminated until successful completion of the sync point operation. If any target server responds with rollback, the conversations are not allowed to be terminated.

To terminate a conversation without requiring the full sync point operation, the sync point forget command may be sent to a target SYNCPTMGR to terminate a released conversation that is not participating in the current unit of work. The sync point new unit of work command must not have been sent since the last sync point operation. The *release* parameter must be set to TRUE.

A sync point request to commit command sent to the resync server may require one or more SYNCLOG command data objects, one for each participating server in the unit of work who replied request to commit to the sync point prepare command. If no participants returned request to commit, no SYNCLOG command object is sent.

The *forget* parameter on the sync point committed command is used to specify the type of forget to be returned by the target SYNCPTMGR. If TRUE, the sync point forget reply data is returned. Otherwise, no reply is returned. The next reply from the target server implies the forget for the unit of work.

If the connection is to be reused at the completion of the sync point operation, the release connection instance variable (*rlsconv*) is set to REUSE on the sync point prepare or rollback command or for the case of the resync server, the sync point request to commit command. For the rollback command, the forget instance variable must be set to true to force the target server to generate a reply.

The connection cannot be reused until the completion of the sync point operation. If any target server responds with *rlsconsset* set to FALSE, the connection is not allowed to be reused until the application terminates. If set to REUSE, the target server can return a set of SQLSTT objects to re-establish the application execution environment for the next transaction.

### XA Support

If the XAMGR is in use on the source system, then the SYNCCTL is used to:

- New Unit of Work  
Start/Register a transaction and associate the XID with the XAMGR protected connection.
- End Transaction  
End the association of a transaction with the current XID. Note that this does not commit or rollback the transaction branch; however, the connection can now be associated with a different transaction branch.
- Prepare Transaction  
Prepare to commit the transaction with XID. Note that the XID need not be the one currently being executed on the connection, but has to be an XID that has been ended.
- Commit Transaction  
Commit the transaction with XID. Note that the XID need not be the one currently being executed on the connection, but has to be an XID that has been ended.
- Rollback Transaction  
Rollback the transaction with XID. Note that the XID need not be the one currently being executed on the connection, but has to be an XID that has been ended.
- Indoubt List  
Request list of XIDs that have been prepared or heuristically commit/rolled back, from the target XAMGR.

- Forget Transaction

Forget a heuristically completed XID.

Replies are expected in all cases and the XAFLAGS must always be sent. See *XAMGROV* (on page 1066) for more information.

**DSS Carrier: RQSDSS**

### Target System Processing

#### SYNCPTMGR Support

When log information is requested, the SYNCPTMGR responds with a SYNCLOG reply object, representing the target SYNCPTMGR log.

When a sync point new unit of work command, a sync point committed command with forget set to FALSE, or a sync point forget command is received, no reply is expected unless an error is detected processing the request.

All other sync point commands require the target SYNCPTMGR to respond with an SYNCPRD reply object.

If the *release* parameter was set to TRUE, the target server can terminate the conversation if at the completion of the sync point operation the decision was to commit the unit of work.

See the SYNCPTMGR state table in *SYNCPTOV* (on page 944) for information.

Any sync point reply data returns reuse, the target server can return a set of SQLSTT objects. These objects are sent to the target server on the EXCSQLSET command prior to executing the next transaction for the application on possibly a different connection. If the sync point rollback operation requests connection reuse, a sync point forget reply is required.

#### XA Support

The target XAMGR process the SYNCCTL requests to perform the following operations on the RDB:

- New Unit of Work
  - Start or resumption of a suspended transaction with the target RDB.
- End Transaction
  - End association of the current XID with the RDB. Note that this does not commit or rollback the transaction branch; however, the connection can now be associated with a different transaction branch.
- Prepare Transaction
  - Prepare to commit the transaction with XID. Note that the XID need not be the one currently being executed on the connection, but has to be an XID that has been ended.
- Commit Transaction
  - Commit the transaction with XID. Note that the XID need not be the one currently being executed on the connection, but has to be an XID that has been ended.
- Rollback Transaction

Rollback the transaction with XID. Note that the XID need not be the one currently being executed on the connection, but has to be an XID that has been ended.

- Indoubt List

Request list of XIDs that have been prepared or heuristically commit/rolled back, from the RDB and return it to the source XAMGR.

- Forget Transaction

Forget a heuristically completed XID.

Replies are required in all cases. See *XAMGROV* (on page 1066) for more information.

clsvar	NIL	
insvar	CLASS INSTANCE VARIABLES	
length	*	
class	X'1055'	
forget	INSTANCE_OF	FORGET - Forget Unit of Work
	LENGTH	2
	OPTIONAL	SYNCPTMGR
	NOTE	For SYNCPTMGR, this parameter must be specified if synctype is:  X'03' - Unit of work committed X'04' - Rollback unit of work X'06' - Forget unit of work  Otherwise, this parameter must not be present.  For XAMGR, this parameter is not valid and should not be present.
ENUVAL	NOTE	X'F1' - TRUE - True State
	NOTE	X'F1' - Forget reply required Required if release was set on SYNCCTL synctype (prepare).
ENUVAL	NOTE	X'F0' - FALSE - False State
	NOTE	X'F0' - Next reply implies forget.
	DFTVAL	X'F1' - TRUE - True State
	MINLVL	SYNCPTMGR - Level 5
rlsconv	INSTANCE_OF	RLSCONV - Release Connection
	OPTIONAL	SYNCPTMGR
	NOTE	For SYNCPTMGR, this parameter can only specified if the synctype is:  X'01' - Prepare to commit X'04' - Rollback unit or work X'05' - Request to commit X'06' - Forget and terminate a connection  Otherwise, the parameter should not be present.  For XAMGR, this parameter can be specified on all synctypes except:

	MINLVL	X'09' - New unit of work SYNCPTMGR - Level 5 XAMGR - Level 7
synctype	INSTANCE_OF REQUIRED CMDTRG NOTE	SYNCTYPE - Sync Point Operation Type XAMGR and SYNCPTMGR  For XAMGR, following synctypes are not valid and should not be present:  X'00' - None X'02' - Migrate resync responsibility X'08' - Request log information X'0A' - Migrated resync responsibility  For SYNCPTMGR, following synctypes are not valid and should not be present:  X'0B' - End association X'0C' - Return list of XIDs
	MINLVL	SYNCPTMGR - Level 5 XAMGR - Level 7
timeout	INSTANCE_OF OPTIONAL IGNORABLE NOTE	TIMEOUT - Time Out XAMGR  For XAMGR, this parameter can only be specified if synctype is:  X'09' - New unit of work  and XAFLAGS is TMNOFLAGS. Otherwise, this parameter should not be present.  For SYNCPTMGR, this parameter is not valid and should not be present.
	DFTVAL MINLVL	0 ms (Do not time out) XAMGR - Level 7
uowid	INSTANCE_OF OPTIONAL NOTE	UOWID - Unit of Work Identifier SYNCPTMGR For SYNCPTMGR, this parameter must be specified if synctype is:  X'06' - Forget unit of work X'09' - New unit of work  Otherwise, the parameter should not be present. This parameter should not be specified for <i>SNA LU6.2 protected connections</i> .  For XAMGR, this parameter is not valid and should not be present.
	MINLVL	SYNCPTMGR - Level 5
xafflags	INSTANCE_OF	XAFLAGS - XA flags

	REQUIRED NOTE	XAMGR Parameter only valid for XAMGR; otherwise, should not be present.
	DFTVAL MINLVL	X'00000000' - TMNOFLAGS - No flags XAMGR - Level 7
xid	INSTANCE_OF REQUIRED OPTIONAL NOTE	XID - Global Transaction Identifier XAMGR SYNCPTMGR For SYNCPTMGR, this parameter can only be specified if synctype is:  X'09' - New unit of work  Otherwise, should not be present.
	MINLVL	SYNCPTMGR - Level 7 XAMGR - Level 7
xidshr	INSTANCE_OF OPTIONAL NOTE	XIDSHR - Share Recoverable Resources SYNCPTMGR For SYNCPTMGR, this parameter can only be specified if synctype is:  X'09' - New unit of work and XID was also specified.  For XAMGR, this parameter is not valid and should not be present.
	ENUVAL NOTE	X'00' - Partial Sharing X'00' - Share recoverable resources and locks when XID matches exactly for a set of DRDA protected connections.
	ENUVAL NOTE	X'01' - Complete Sharing X'01' - Share recoverable resources and locks when <i>gtrid</i> part of the XID matches for a set of DRDA protected connections.
	DFTVAL MINLVL	X'00' - Partial Sharing SYNCPTMGR - Level 7
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>cmddta</b>		<b>COMMAND OBJECTS</b>
X'106F'	INSTANCE_OF OPTIONAL REPEATABLE NOTE	SYNCLOG - Sync Point Log SYNCPTMGR  For SYNCPTMGR, must be sent if there are other participants in the unit of work and if synctype is:  X'01' - Prepare to commit X'05' - Request to commit  Otherwise, this command data must not be present.

		For XAMGR, this object is not valid.
<b>rpydta</b>	SQLSTT	<b>REPLY OBJECTS</b>
X'1248'	INSTANCE_OF REQUIRED OPTIONAL NOTE	SYNCCRD - Sync Point Control Reply XAMGR SYNCPTMGR For XAMGR, SYNCCRD is sent in response to all SYNCCTL commands.  For SYNCPTMGR, SYNCCRD is sent in response to a SYNCCTL command if the synctype is:  X'01' - Prepare to commit X'02' - Migrate to commit X'03' - Committed and forget parameter set to TRUE X'05' - Request to commit X'06' - Forget and forget parameter set to TRUE
X'106F'	INSTANCE_OF OPTIONAL NOTE	SYNCLOG - Sync Point Log SYNCPTMGR For SYNCPTMGR, SYNCLOG is returned only when synctype is set to X'08' (request log).  For XAMGR, this object is not valid.
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported
X'2414'	INSTANCE_OF REPEATABLE OPTIONAL NOTE	SQLSTT - SQL Statements  Returned if RLSCONV is set to X'F2' (REUSE) on the SYNCCTL command. Otherwise, this object is not returned.
	MINLVL	7

**SEE ALSO**

<b>insvar</b>	<i>PRCCNVCD</i> (on page 621) <i>SYNCCRD</i> (on page 913)
<b>Semantic</b>	<i>AGENT</i> (on page 61) <i>RSYNCMGR</i> (on page 787) <i>SYNCPTOV</i> (on page 944) <i>TIMEOUT</i> (on page 1019) <i>XAMGR</i> (on page 1063) <i>XAMGROV</i> (on page 1066)

NAME

SYNCLOG — Sync Point Log

DESCRIPTION (Semantic)

**Dictionary** QDDBASD  
**Codepoint** X'106F'  
**Length** \*  
**Class** CLASS  
**Sprcls** Collection

Sync Point Log (SYNCLOG) defines log and resynchronization information used by a SYNCPTMGR. This information is exchanged between source and target SYNCPTMGR to determine the log used during the sync point operation. The connection token (CNNTKN) identifies an instance of a server associated with the connection.

DSS Carrier: OBJDSS

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'106F'	
cnntkn	INSTANCE_OF REQUIRED	CNNTKN - Connection Token
ipaddr	INSTANCE_OF NOTE MTLEXC	IPADDR - TCP/IP Address Resync address for TCP/IP conversations. X'11E9' - SNAADDR - SNA Address
logname	INSTANCE_OF REQUIRED	LOGNAME - Log Name
logstmp	INSTANCE_OF REQUIRED	LOGSTMP - Log Timestamp
snaaddr	INSTANCE_OF NOTE MTLEXC	SNAADDR - SNA Address Resync address for SNA conversations. X'11E8' - IPADDR - TCP/IP Address
rdbnam	INSTANCE_OF REQUIRED	RDBNAM - Relational Database Name
tcphost	INSTANCE_OF OPTIONAL NOTE MTLEXC	TCPHOST - TCP/IP Domain Qualified Host Name  TCP/IP host name of CMNTCPIP. X'11E9' - SNAADDR - SNA Address
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

SEE ALSO

**cmddta** SYNCCTL (on page 915)  
 SYNCRSY (on page 982)  
**rpydta** SYNCCTL (on page 915)



**Semantic**

*SYNCCTL* (on page 915)  
*SYNCPTOV* (on page 944)  
*SYNCRSY* (on page 982)

NAME

SYNCMNBK — LU 6.2 Sync Point Communications Backout

DESCRIPTION (Semantic)

Dictionary QDDTTRD

Length \*

Class HELP

LU 6.2 Sync Point Communications Backout (SYNCMNBK) illustrates normal use of the BACKOUT verb that the source communications manager (SCM) issues. The underlying communications facility is SNA LU 6.2 using the sync point tower. This information has been extracted from the *SNA LU 6.2 Reference: Peer Protocols* (SC31-6808, IBM).

A sample protocol sequence is shown in Figure 3-81. SNA LU 6.2 functions provide so many capabilities that it is impossible to show all the possible sequences. Notes describing key points in the sequence follow the sequence diagram.

The following assumptions have been made for the sample protocol sequence:

- A conversation has been successfully established between the SCM and the target communications manager (TCM) as described in *SYNCMNI* (on page 931) with `ALLOCATE(SYNC_LEVEL(SYNCPT))`.
- No error situation occurs.

The phrase *other protected resource managers* (represented by *Other Prtd Mgrs* in the figure) means those managers on the local system which protect their resources by using the two-phase commit process. For instance, the source protected resource managers might be the SQLAM and the SCM, and the target protected resource managers might include the SQLAM, the TCM, and the SNA LU 6.2.

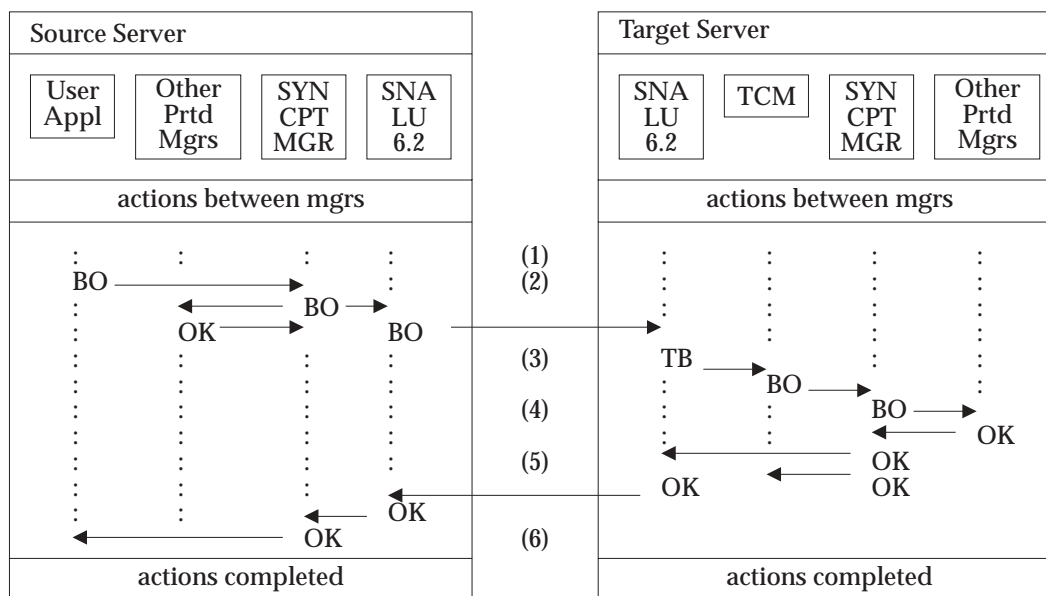


Figure 3-81 Backout Flows Between Managers (SYNCMNBK) Protocol

**Legend**

BO	Backout
OK	Positive response to the backout
TB	Take backout

**Figure Notes**

1. The TCM issues a `RECEIVE_AND_WAIT` verb to receive the next RQSDSS from the SCM.  
The `RESOURCE` parameter identifies the conversation resource to be used. The `FILL` parameter specifies that the TCM wants to receive a single logical record. The `LENGTH` parameter specifies the total length of the buffer space available at the `DATA` parameter location. The buffer space should be large enough to receive the largest anticipated RQSDSS from the SCM.
2. The source application program requests a backout of the unit of work (UOW). The source `SYNCPTMGR` notifies the SNA LU 6.2 communications facilities to backout. The `SYNCPTMGR` notifies the `SQLAM` (and any other manager that is registered as a protected resource manager with the `SYNCPTMGR`) to backout. The order in which the `SYNCPTMGR` notifies the protected resource managers is not defined in DDM architecture.
3. On the target system, the `RECEIVE_AND_WAIT` verb is completed and the `RETURN_CODE` is set to `TAKE_BACK_OUT`.  
The TCM issues a `BACKOUT` verb to the target `SYNCPTMGR` which performs the backout processing.
4. The target `SYNCPTMGR` notifies the protected resource managers to backout. The notified protected resource managers may include additional SNA LU 6.2 protected resource managers if there are other conversations in the UOW besides the conversation that included the backout request. When the other protected resource managers complete their backout processing, they notify the `SYNCPTMGR`.
5. When the `SYNCPTMGR` receives notification from the other protected resource managers, the `SYNCPTMGR` notifies the source system that backout processing is complete at the target system. This is accomplished by notifying the SNA LU 6.2 facilities. A positive response to the backout flows to the source system. In addition, the target `SYNCPTMGR` posts a positive response to the TCM for the backout verb issued in note (3).
6. When the source `SYNCPTMGR` receives the `OK` responses from the protected resource managers, a positive response to the backout is given to the application program.

**SEE ALSO**

<b>Semantic</b>	<i>CMNSYNCPT</i> (on page 202)
	<i>SYNCMNBK</i> (on page 924)

NAME

SYNCMNCM — LU 6.2 Sync Point Communications Two-Phase Commit

DESCRIPTION (Semantic)

Dictionary QDDTTRD

Length \*

Class HELP

LU 6.2 Sync Point Communications Two-Phase Commit (SYNCMNCM) illustrates normal use of the SYNCPT verb the source communications manager (SCM) issues. The underlying communications facility is SNA LU 6.2 using the sync point tower. This information has been extracted from the *SNA LU 6.2 Reference: Peer Protocols* (SC31-6808, IBM).

A sample protocol sequence is shown in Figure 3-82. SNA LU 6.2 functions provide so many capabilities that it is impossible to show all the possible sequences. Notes describing key points in the sequence follow the sequence diagram.

The following assumptions have been made for the sample protocol sequence:

- A conversation has been successfully established between the SCM and the target communications manager (TCM) as described in *SYNCMNI* (on page 931) with `ALLOCATE(SYNC_LEVEL(SYNCPT))`.
- No error situation occurs.

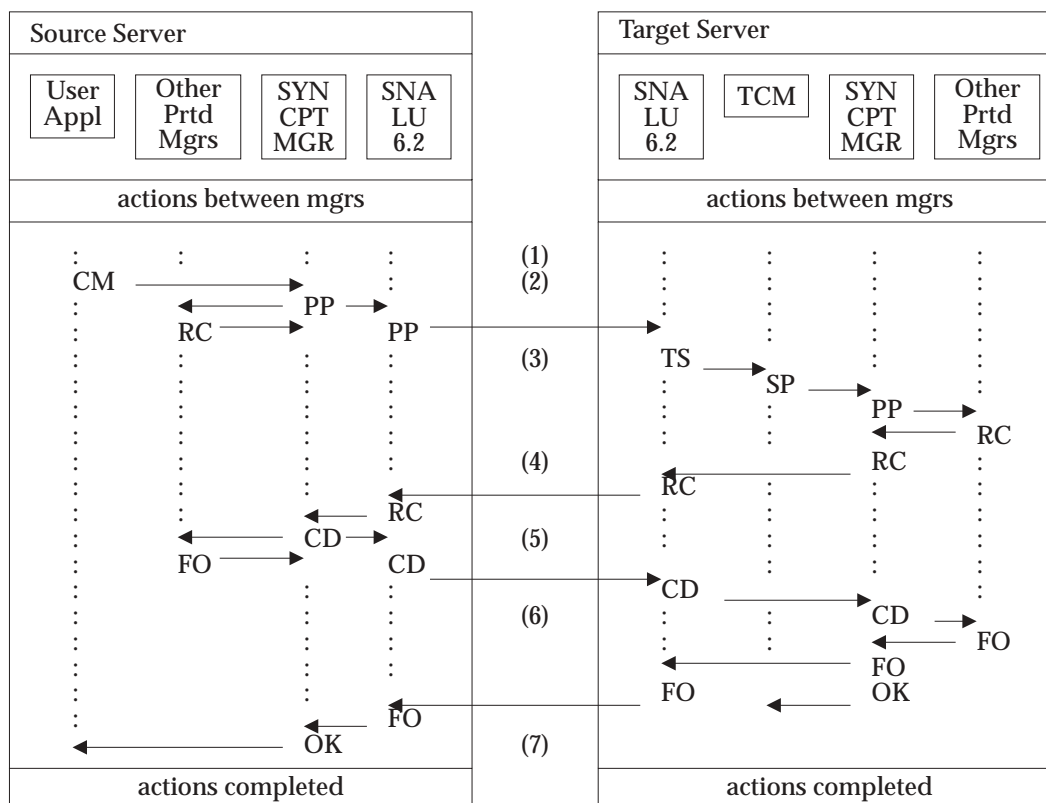


Figure 3-82 Two-Phase Commit Flows Between Managers (SYNCMNCM) Protocol

**Legend**

CM	Commit request
PP	Prepare to commit
OK	Positive response
TS	Take Syncpt
SP	SYNCPT verb
RC	Request commit
CD	Committed
FO	Forget message

**Figure Notes**

1. The TCM issues a RECEIVE\_AND\_WAIT verb to receive the next RQSDSS from the SCM.  
The RESOURCE parameter identifies the conversation resource to be used. The FILL parameter specifies that the TCM wants to receive a single logical record. The LENGTH parameter specifies the total length of the buffer space available at the DATA parameter location. The buffer space should be large enough to receive the largest anticipated RQSDSS from the SCM.
2. The source application program requests the SYNCPTMGR to commit the unit of work (UOW). The source SYNCPTMGR notifies the SNA LU 6.2 communications facilities to prepare to commit and notifies the SQLAM (and other protected resource managers registered with the SYNCPTMGR) to prepare to commit. The source communications facility sends the SNA LU 6.2 prepare message to the target system. The local protected resource managers respond to the source SYNCPTMGR with the *Request Commit* message.
3. On the target system, the RECEIVE\_AND\_WAIT verb is completed and the WHAT\_RECEIVED parameter is set to TAKE\_SYNCPT.  
The TCM issues a SYNCPT verb to the target SYNCPTMGR which begins the commit processing. The SYNCPTMGR prepares the protected resources to commit.
4. The SYNCPTMGR sends the SNA LU 6.2 request commit message to the source system.
5. The source SYNCPTMGR collects the request commit messages from the SNA LU 6.2 communications facilities and the other protected resource managers. The source SYNCPTMGR then commits the unit of work by requesting that all of the resources commit. This causes an SNA LU 6.2 committed message to be sent to the target system.
6. The target SYNCPTMGR requests that the local resources commit the unit of work and causes an SNA LU 6.2 forget message to be sent to the source system. In addition, the target SYNCPTMGR posts a positive response to the TCM for the SYNCPT verb issued in note (3).
7. When the source SYNCPTMGR receives the FO responses from the protected resource managers, a positive response to the commit is given to the application program.

**SEE ALSO**

<b>Semantic</b>	<i>CMNSYNCPT</i> (on page 202)
	<i>SYNCMNT</i> (on page 935)

## NAME

SYNCFMFL — LU 6.2 Sync Point Communications Failure

## DESCRIPTION (Semantic)

**Dictionary** QDDTTRD**Length** \***Class** HELP

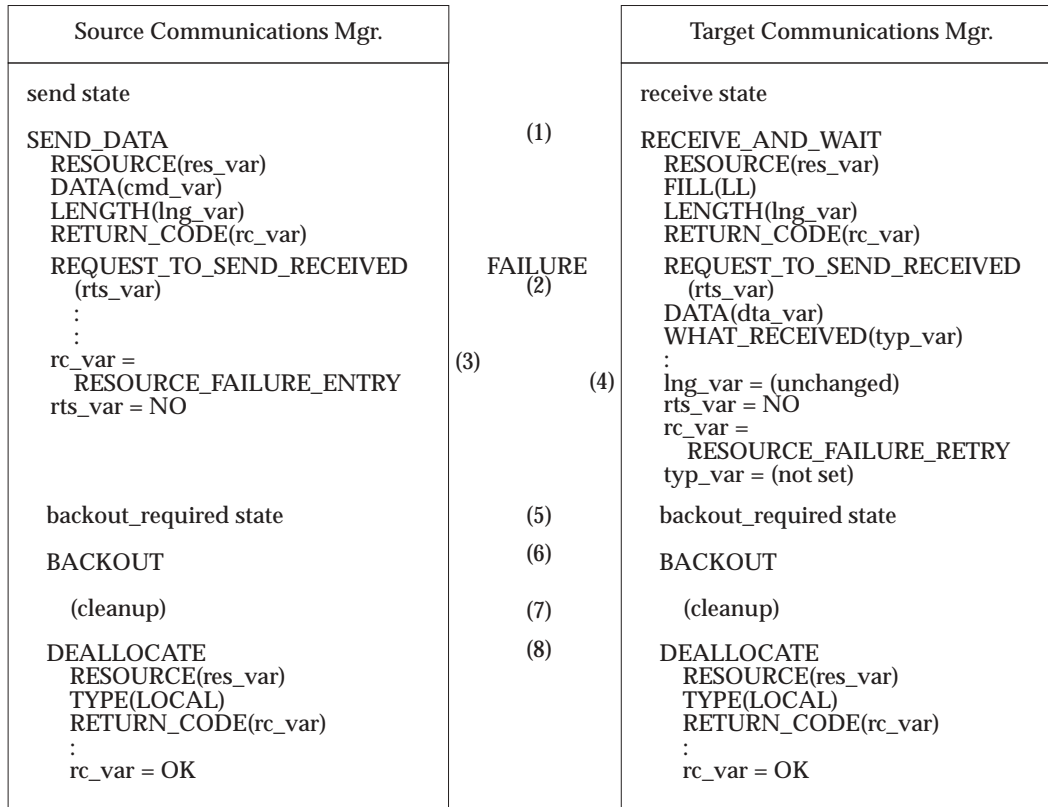
LU 6.2 Sync Point Communications Failure (SYNCFMFL) illustrates the communications sequence that occurs when communications between the source communications manager (SCM) and the target communications manager (TCM) prematurely terminates. A communications failure can be caused by an SNA LU 6.2 session protocol error, an SNA LU 6.2 session outage, a communications line failure, a modem failure, a remote system failure, or by other failures. Thus SCM and TCM cannot communicate. Do not confuse communications failures with DDM-detected errors that result in a reply message. See the SNA manuals for detailed information about SNA communications failures.

The basic sequence for handling a communications failure is for both the SCM and the TCM to deallocate their SNA conversation and perform any required cleanup. Figure 3-83 (on page 929) illustrates this sequence.

The sequence illustrated is only a sample. SNA LU 6.2 functions provide so many capabilities that it is impossible to show all the possible sequences. Notes describing key points in the sequence follow the sequence diagram.

The following assumptions have been made for the sample protocol sequence:

- A conversation is successfully established between the SCM and the TCM. (A successful initiation of a conversation is described in *SYNCFMFI* (on page 931).)
- The SCM and the TCM both perform synchronous sends and receives.
- The application program is within a unit of work (UOW); that is, a commit or backout is not pending. If the application program has issued a commit or backout and a failure occurs, see *RESYNOVR* (on page 764) and the *SNA LU 6.2 Reference: Peer Protocols* (SC31-6808, IBM).



**Figure 3-83** Communications Failure Between Source and Target (SYNCMNFL)

**Figure Notes**

1. The SCM and TCM are passing RQSDSS and RPYDSS structures to each other. The SCM issues a SEND\_DATA verb to send a RQSDSS to the TCM. The TCM issues a RECEIVE\_AND\_WAIT verb to receive the next RQSDSS from the SCM.
2. A communications failure occurs. The communications resource (in this case, communications line) is broken and the source and target systems cannot communicate.
3. The SEND\_DATA verb completes its operation on the source system with a return code that indicates that the communications resource has failed. This tells the SCM that communications with the TCM are no longer possible with the currently allocated SNA LU 6.2 conversation.  
  
The SNA LU 6.2 communications facility sets the RETURN\_CODE parameter to RESOURCE\_FAILURE\_RETRY and the REQUEST\_TO\_SEND\_RECEIVED parameter to NO. The RESOURCE\_FAILURE\_RETRY return code indicates that attempts to reestablish communications may be successful.
4. The RECEIVE\_AND\_WAIT verb completes its operation on the target system with a return code that indicates that the communications resource has failed. This tells the TCM that communications with the SCM are no longer possible with the currently allocated SNA LU 6.2 conversation.  
  
The SNA LU 6.2 communications facility does not change the LENGTH parameter; it sets the REQUEST\_TO\_SEND\_RECEIVED parameter to NO and the RETURN\_CODE

parameter to RESOURCE\_FAILURE\_RETRY. The RESOURCE\_FAILURE\_RETRY return code indicates that attempts to reestablish communications may be successful.

5. The SCM and TCM are placed in the BACKOUT\_REQUIRED state; they each issue a BACKOUT verb to their respective SYNCPTMGRs. Although only one SYNCPTMGR is allowed per server, multiple SYNCPTMGRs may be participating in the current UOW. If so, the BACKOUT verb is propagated to each of them.
6. Each SYNCPTMGR coordinates the BACKOUT operation on the protected resources.
7. The SCM notifies the source agent of the failure and performs any required cleanup functions. This may include cleaning up internal tables and control blocks, logging the failure, and other cleanup functions.

The TCM notifies the target agent of the failure and performs any required cleanup functions. Upon notification of the failure, the target agent notifies the other server managers to perform cleanup functions. For servers that support relational databases, these include:

- Terminating any target SQLAM manager instance
  - Performing any additional cleanup functions required
8. The SCM issues a DEALLOCATE verb to deallocate the SNA LU 6.2 conversation.
    - The RESOURCE parameter specifies the SNA LU 6.2 conversation resource. The TYPE parameter specifies LOCAL because to perform the deallocate function there are no communications with the target system.
    - The DEALLOCATE verb completes with a RETURN\_CODE of OK. The SCM is now completely disassociated from the SNA LU 6.2 communications facility on the source system.

The TCM issues a DEALLOCATE verb to deallocate the SNA LU 6.2 conversation.

- The RESOURCE parameter specifies the SNA LU 6.2 conversation resource. The TYPE parameter specifies LOCAL because to perform the deallocate function there are no communications with the source system.
- The DEALLOCATE verb completes with a RETURN\_CODE of OK. The TCM is now completely disassociated from the SNA LU 6.2 communications facility on the target system.

#### SEE ALSO

**Semantic**            *CMNSYNCPT* (on page 202)  
                           *SYNCFMFL* (on page 935)



**NAME**

SYNCMNI — LU 6.2 Sync Point Communications Initiation

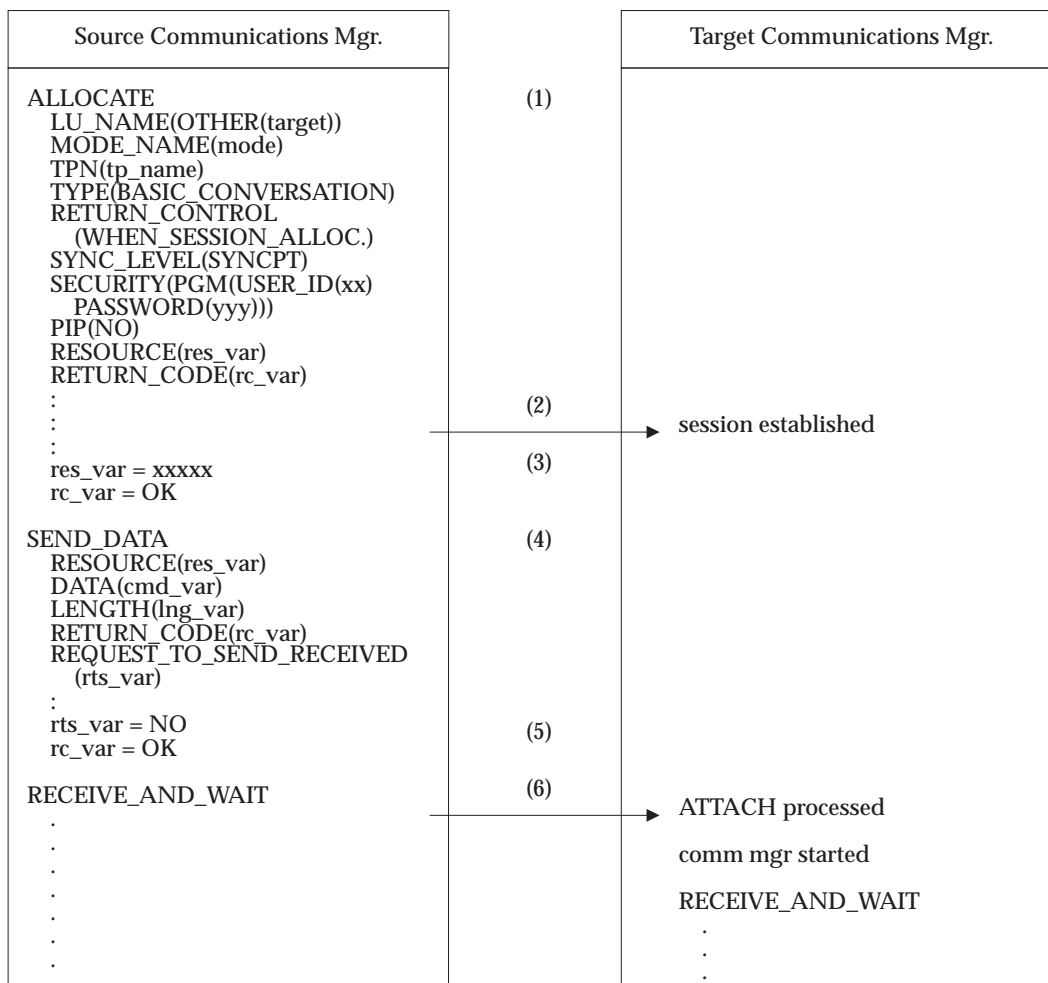
**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

LU 6.2 Sync Point Communications Initiation (SYNCMNI) illustrates the use of SNA sync point communications facilities to initiate source-to-target communications. The SNA LU 6.2 sync point communications facility is responsible for the SNA LU 6.2 session initiation. More information about SNA LU 6.2 session initiation is in the SNA LU 6.2 manuals.

A sample protocol sequence is shown in Figure 3-84 (on page 932). SNA LU 6.2 functions provide so many capabilities that it is impossible to show all the possible sequences. Notes describing key points in the sequence follow the sequence diagram.

The following assumptions have been made for the sample protocol sequence:

- No conversation exists between the source communications manager (SCM) and the target communications manager (TCM).
- The SCM and TCM both can perform synchronous sends and receives.
- Both systems support SYNC\_LEVEL(SYNCPT).
- The session limit between the source and target logical units (LUs) has not been reached.
- No error situations occur.



**Figure 3-84** Communications Initiation by Source System (SYNCMNI)

**Figure Notes**

1. The SCM establishes communications with the TCM by issuing an ALLOCATE verb to the SNA LU 6.2 communications facility. The ALLOCATE verb contains the information the SNA LU 6.2 needs to determine where the TCM is located (LU\_NAME), the type of SNA LU 6.2 services being used (MODE\_NAME), the type of verb interface the SCM uses with the SNA LU 6.2 (TYPE), and the synchronization level (SYNC\_LEVEL).

In addition to these communications parameters, the ALLOCATE verb contains the identification of the transaction program being initiated (TPN), parameter data that the transaction program (PIP) might require, and the security information for verifying the user (SECURITY).

The SECURITY parameter may specify NONE, SAME, or PGM. If NONE is specified, some target servers that require user identification may reject the allocation before allowing access to their resources. DDM allows the use of *already verified* APPC security functions but does not require that a target server instance support them. In this illustration, the USER\_ID and PASSWORD of the requester are sent to the target system for verification.

The ALLOCATE verb only requests that a conversation and session with the remote location be assigned to the SCM.

2. The SNA LU 6.2 communications facility tries to find an existing, unused session with the remote location; or, if none are available, the SNA LU 6.2 attempts to establish a session. More information on this process is in the SNA LU 6.2 manuals.
3. When the SNA LU 6.2 communications facility has successfully assigned a conversation and session to the SCM, the RETURN\_CODE variable is set to OK. The RESOURCE variable is set to the resource identifier for the assigned conversation. Control is returned to the SCM (because the RETURN\_CONTROL parameter specified WHEN\_SESSION\_ALLOCATED). The SCM has not yet actually communicated with the TCM.
4. The SCM issues a SEND\_DATA verb to the SNA LU 6.2 communications facility to cause the first DDM command (RQSDSS) to be sent to the TCM. This first command must be an exchange server attributes (EXCSAT) command to determine the server and manager levels of the target server.

The RESOURCE parameter identifies the conversation resource to be used. This parameter has the same value as that returned on the ALLOCATE verb. The DATA parameter specifies the location of the data (RQSDSS) to be sent. The LENGTH parameter gives the total length of the data at the DATA location.

5. The SNA LU 6.2 communications facility accepts the data which is queued for output. It is not sent to the TCM at this time.

Once the SNA LU 6.2 communications facility has successfully enqueued the data, the RETURN\_CODE parameter is set to OK, and control is returned to the SCM process. In this example, REQUEST\_TO\_SEND\_RECEIVED is set to NO.

6. The SCM issues a RECEIVE\_AND\_WAIT verb to receive the RPYDSS or OBJDSS from the RQSDSS it just sent. This verb causes the SNA LU 6.2 communications facility to transmit all of the data in its transmit buffers and to send the TCM the *right-to-send* indicator.

The contents of the transmission are:

- A bracket bid, chaining information, and a change direction indication
- An ATTACH header to cause the transaction program (the TCM transaction program) to initiate on the target system and attach to the same conversation as the SCM

The transaction program name (TPN) can be any valid TPN that invokes a DDM communications manager. The term CMNSYNCPT contains more information about the TPN.

The ATTACH header also contains the user security information.

- Data

This is the RQSDSS the SCM built.

At this point, communications have been established with the target system, the TCM process has been initiated, and DDM commands have begun to flow.

#### SEE ALSO

<b>Semantic</b>	<i>APPSRCCD</i> (on page 79)
	<i>APPSRCCR</i> (on page 86)
	<i>APPSRCER</i> (on page 91)
	<i>APPTRGER</i> (on page 95)
	<i>CMNSYNCPT</i> (on page 202)

*SYNCMNBK* (on page 924)  
*SYNCMNCM* (on page 926)  
*SYNCMNFL* (on page 928)  
*SYNCMNT* (on page 935)

**NAME**

SYNCMNT — LU 6.2 Sync Point Communications Termination

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

LU 6.2 Sync Point Communications Termination (SYNCMNT) illustrates normal communications termination by the source communications manager (SCM).

Under normal circumstances, only the (SCM) can terminate the conversation between the SCM and the target communications manager (TCM). See *SYNCMNFL* (on page 928) for abnormal circumstances. The SCM should terminate the conversation only when the source system has completed all its work with the target system.

A sample protocol sequence is shown in Figure 3-85 (on page 936). SNA LU 6.2 functions provide so many capabilities that it is impossible to show all the possible sequences. Notes describing key points in the sequence follow the sequence diagram.

The following assumptions have been made for the sample protocol sequence:

- A conversation exists between the SCM and the TCM. See *SYNCMNI* (on page 931) for more information about initiating conversations.
- No error situation occurs.

The SNA LU 6.2 session may or may not be terminated as a result of a normal sync point communications conversation sequence. The SNA LU 6.2 facility, not the DDM communications manager controls termination.

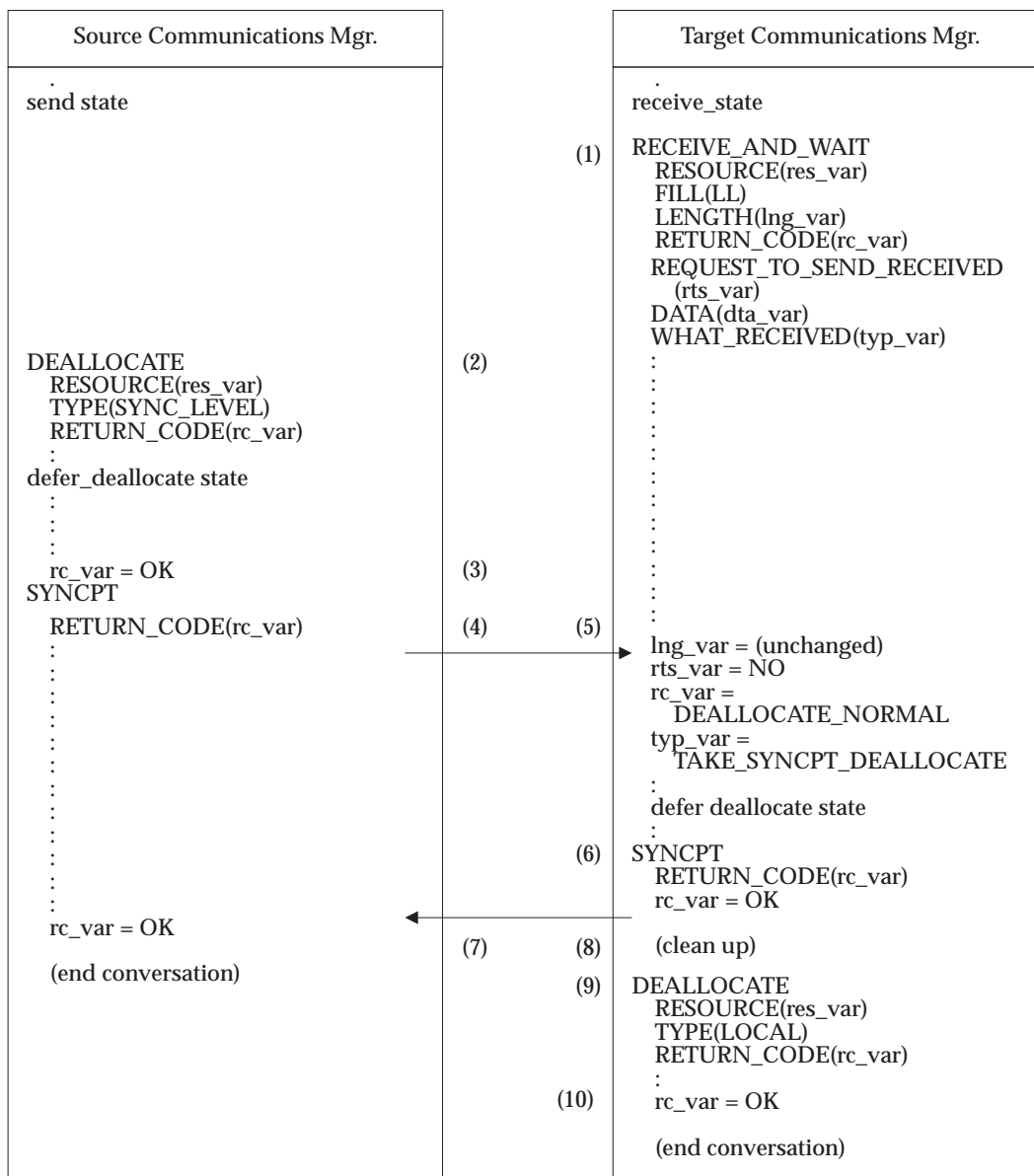


Figure 3-85 Normal Communications Termination (SYNCMNT) Protocol

**Figure Notes**

1. The TCM issues a RECEIVE\_AND\_WAIT verb to receive the next RQSDSS from the SCM.  
The RESOURCE parameter identifies the conversation resource to be used. The FILL parameter specifies that the TCM wants to receive a single logical record (LL). The LENGTH parameter specifies the total length of the buffer space available at the DATA parameter location. The buffer space should be large enough to receive the largest anticipated RQSDSS from the SCM.
2. The source system finishes using the remote data facilities of the target system. The SCM then issues a DEALLOCATE verb to deallocate the SNA LU 6.2 conversation.

The RESOURCE parameter identifies the SNA LU 6.2 conversation being deallocated and the TYPE parameter specifies that the deallocation process uses the SYNC\_LEVEL of the conversation.

The source system is placed in the DEFER\_DEALLOCATE state.

3. The DEALLOCATE verb completes its operation with a RETURN\_CODE of OK.
4. The SCM issues a SYNCPT verb. Two-phase commit protocols occur (see *SYNCMNCM* (on page 926) for a description of these protocols).
5. The target system receives the deallocate/detach indications and causes the RECEIVE\_AND\_WAIT verb to complete its operation. Control is returned to the TCM.

The target SNA LU 6.2 communications facility:

- Does not set the LENGTH parameter.
- Sets the REQUEST\_TO\_SEND\_RECEIVED parameter to NO.
- Sets the RETURN\_CODE parameter to DEALLOCATE\_NORMAL.
- Sets the WHAT\_RECEIVED parameter to TAKE\_SYNCPT\_DEALLOCATE.

The source SNA LU 6.2 facility transmits available queued data along with the deallocation request to the TCM. This example assumes that no queued data is available to be sent to the TCM.

The transmission contents (not illustrated) are possibly:

- Any remaining data (SNA FLUSH verb)
  - Chaining information and an end bracket (deallocate/detach) indication
  - An LUSTAT command with a sense code of 0006; the LUSTAT is simply a carrier for the information listed above
6. The TCM issues a SYNCPT verb and it completes with a return code of OK.
  7. The SYNCPT verb the SCM issues completes its operation with a RETURN\_CODE of OK. The SCM is now completely disassociated from the SNA LU 6.2 Sync Point communications facility on the source system.
  8. The TCM notifies the target agent which notifies the other server's managers to perform any required cleanup functions.
  9. The TCM deallocates its SNA LU 6.2 communications conversation.

The RESOURCE parameter specifies the SNA LU 6.2 conversation resource. The TYPE parameter specifies LOCAL because there are no communications with the source system to perform the deallocate function.

10. The DEALLOCATE verb completes its operation with a RETURN\_CODE of OK. The TCM is now completely disassociated from the SNA LU 6.2 communications facility on the target system.

**SEE ALSO**

**Semantic**

*CMNSYNCPT* (on page 202)



**NAME**

SYNCPTMGR — Sync Point Manager

**DESCRIPTION (Semantic)****Dictionary** QDDBASD**Codepoint** X'14C0'**Length** \***Class** CLASS**Sprcls** MANAGER - Resource Manager

Sync Point Manager (SYNCPTMGR) is a manager object of DDM that coordinates resource recovery of the units of work (UOW) associated with recoverable resources in multiple DDM servers. The SYNCPTMGR supports the two-phase commit process. If an UOW is across multiple servers and the resources are protected, then the SYNCPTMGR must be used.

A sync point manager maintains consistency in changes made to protected resources. The primary functions of a sync point manager include, but are not limited to, the following:

1. Keeping track of and logging UOW state information (see the references below for information on UOW state)
2. Keeping track of and logging all local protected resource manager (PRM) names that are involved with a unit of work
3. Coordinating the COMMIT and ROLLBACK of all local PRMs
4. Initiating resynchronization protocols for any unit of work that may be in the in-doubt state because of a system or communications failure

For more information, see the *SNA Format and Protocol Reference Manual: Architecture Logic For LU Type 6.2* (SC30-3269, IBM), *SNA LU 6.2 Reference: Peer Protocols* (SC31-6808, IBM), and the DRDA Reference.

The role of the SYNCPTMGR at DDM Level 4 is illustrated in Figure 3-86 (on page 940).

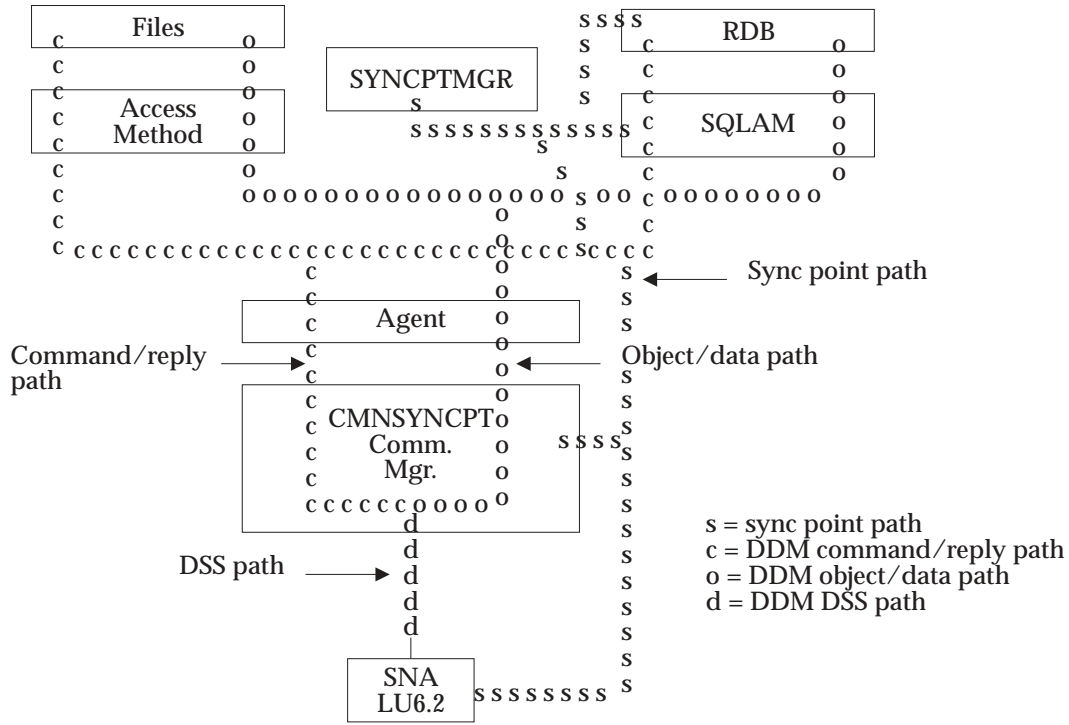


Figure 3-86 Server Paths for SYNCPTMGR at DDM Level 4

The role of the SYNCPTMGR at DDM Level 5 is illustrated in Figure 3-87 (on page 941). Sync point operations flow as DDM commands, objects, and replies using either the TCP/IP or the LU 6.2 communication manager. The agent forwards sync point commands and objects to the SYNCPTMGR which then interfaces to the RDB manager or SQLAM to perform sync point operations. Sync point replies are sent from the SYNCPTMGR to the AGENT in the form of DDM reply and DDM objects which are sent by the AGENT using the communication manager to the remote AGENT.

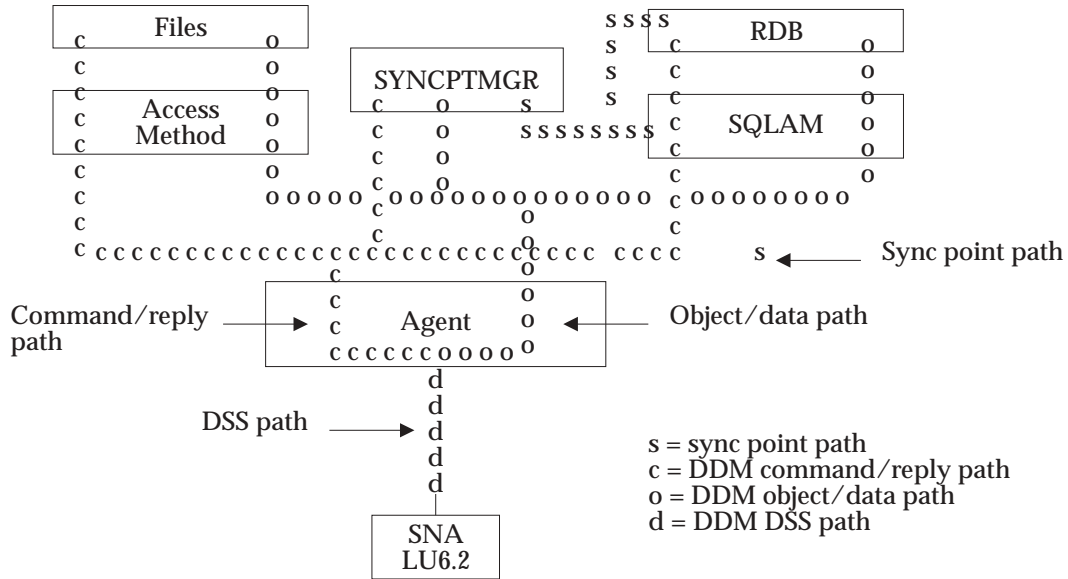


Figure 3-87 Server Paths for SYNCPTMGR at DDM Level 5

The Sync point manager and RDB can share recoverable resources and locks between a set of SYNCPTMGR protected connections that are identified by the same XID value. The XID value is sent during a new unit of request. The RDB uses the XID to group SYNCPTMGR protected connections that have the same value. These groups of protected connections can share recoverable resources between themselves so as to prevent any deadlocks.

**Manager-Level Compatibility**

Table 3-17 illustrates the function of the SYNCPTMGR as it has grown and changed through the levels of DDM architecture.

Table 3-17 Sync Point Manager-Level Compatibility

DDM Levels	1	2	3	4	5	7
SYNCPTMGR				4	5	7
<i>Manager Dependencies</i>						
CMNSYNCPT				4		
RSYNCMGR (required for resync support)					5	7
CMNAPPC					3	3
CMNTCPIP					5	5

SYNCPTMGR Level 4 depends on and uses SNA LU 6.2 communication facilities to coordinate commit. SYNCPTMGR Level 5 uses DDM commands to coordinate commit and can be used with SNA LU 6.2 SYNC\_LEVEL(NONE) or TCP/IP. See SYNCPTOV (on page 944).

clsvar                      NIL

<b>insvar</b>	<b>NIL</b>	
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	
<b>mgrlvl</b>	7	
<b>mgrdepls</b>		<b>MANAGER DEPENDENCY LIST</b>
X'147C'	INSTANCE_OF MGRLVLN NOTE	CMNSYNCPT - SNA LU 6.2 Sync Point Conversational Communications Manager 4 The communications manager must support SNA LU 6.2 SYNC_LEVEL(SYNCPT).
X'14C1'	INSTANCE_OF MGRLVLN NOTE	RSYNCMGR - Resynchronization Manager 5 Required only for resync server support.
X'1444'	INSTANCE_OF MGRLVLN NOTE	CMNAPPC - LU 6.2 Conversational Communications Manager 3 The communications manager must support SNA LU 6.2 SYNC_LEVEL(NONE).
X'1474'	INSTANCE_OF MGRLVLN	CMNTCPIP - TCP/IP Communication Manager 5
X'1403'	INSTANCE_OF MGRLVLN	AGENT - Agent 5
<b>vldattls</b>		<b>VALID ATTRIBUTES</b>
X'0019'	INSTANCE_OF	HELP - Help Text
X'1452'	INSTANCE_OF	MGRNAM - Manager Name
X'0045'	INSTANCE_OF	TITLE - Title

**SEE ALSO**

<b>cmdtta</b>	<i>SYNCCTL</i> (on page 915) <i>SYNCRSY</i> (on page 982)
<b>insvar</b>	<i>MGRLVL</i> (on page 506) <i>SYNCCRD</i> (on page 913) <i>SYNCCTL</i> (on page 915)
<b>mgrdepls</b>	<i>CMNSYNCPT</i> (on page 202) <i>RSYNCMGR</i> (on page 787)
<b>rpydta</b>	<i>SYNCCTL</i> (on page 915)
<b>Semantic</b>	<i>AGENT</i> (on page 61) <i>CMNSYNCPT</i> (on page 202) <i>CNNTKN</i> (on page 220) <i>INHERITANCE</i> (on page 437) <i>LOGNAME</i> (on page 484) <i>LOGTSTMP</i> (on page 485) <i>MANAGER</i> (on page 491) <i>MGRLVLN</i> (on page 509) <i>RDBOVR</i> (on page 740) <i>RESYNOVR</i> (on page 764) <i>RSYNCMGR</i> (on page 787)

*SQLAM* (on page 847)  
*SUBSETS* (on page 902)  
*SYNCCR*D (on page 913)  
*SYNCCTL* (on page 915)  
*SYNCLOG* (on page 922)  
*SYNCMNBK* (on page 924)  
*SYNCMNCM* (on page 926)  
*SYNCMNFL* (on page 928)  
*SYNCPTOV* (on page 944)  
*SYNCRRD* (on page 981)  
*SYNCRSY* (on page 982)  
*SYNCTYPE* (on page 984)  
*UOWSTATE* (on page 1040)

## NAME

SYNCPTOV — Overview for Sync Point Flows

**Semantic (DESCRIPTION)**

**Dictionary** QDDTTRD

**Length** \*

**Class** HELP

This term is an overview of the DDM SYNCPTMGR at Level 5 in conjunction with the DDM RSYNCMGR at Level 5. SYNCPTMGR at Level 5 supports the concept of distributed unit of work. A distributed unit of work is a transaction which accesses and possibly updates protected resources (such as RDBs) at one or more servers. SYNCPTMGR provides for coordination of commit and rollback as well as error recovery among the servers in order to maintain consistency among the protected resources.

SYNCPTMGR at Level 4 relies on SNA LU 6.2 protected conversations in order to provide resource coordination and recovery. SYNCPTMGR at Level 5 uses two new DDM commands (SYNCCTL and SYNCRSY) to provide resource coordination and recovery. SYNCPTMGR at Level 5 is capable of managing either TCP/IP or SNA conversations. SYNCPTMGR Level 4 is restricted to SNA conversations.

SYNCPTMGR Level 5 uses an optimized two-phase commit protocol known as presumed abort (PA). This optimization reduces the number of forced log writes at the servers and reduces the number of network message flows. Only the PA protocol is supported. The DDM command SYNCCTL implements the two-phase commit protocol operations.

SYNCPTMGR Level 5 also allows the use of an optional "implied forget" processing in which the next DDM command acts as an implied acknowledgement that commit processing has completed. This reduces network message flows and allows a source server to initiate another unit of work before all target servers have completed commit processing.

RSYNCMGR Level 5 supports coordinated error recovery (called resynchronization) performed when a server does not know the outcome of a unit of work because an error occurred that prevented two-phase commit from completing. After recovering from the failure, the unit of work is completed by reconnecting and performing resynchronization using the SYNCRSY command.

In some cases a source server may not have the resources required to support error recovery. Error recovery requires a log which must persist over time and deletion of a log causes protected resources to be exposed to data integrity problems. To support two-phase commit for source servers without logs, error recovery responsibilities can be migrated to a target server that supports RSYNCMGR Level 5. In this case, the target server becomes known as the resync server. If a failure occurs during commit, the resync server performs all required logging during the commit operation and performs any required resynchronizations.

SYNCPTMGR Level 7 allows the application to share resource and locks between a set of SYNCPTMGR protected connections on the application server. Each connection is identified by an XID sent on a new uow of work request. The set of SYNCPTMGR protected connections that have the same XID value can share resources and locks between themselves, so as to prevent any deadlocks from occurring. The XIDSHR share parameter allows the application to inform the application server how to identify this set of connections based on the XID.

### Manager-Level Dependencies

1. SYNCPTMGR Level 5 and a RSYNCMGR Level 0 are the minimal manager levels required to support the sync point control command to perform the two-phase commit. The sync point migrate command is not supported.
2. SYNCPTMGR Level 5 and a RSYNCMGR Level 5 are the minimal manager levels required to support the sync point migrate command. The target SYNCPTMGR can act as the resync server for the source server.
3. SYNCPTMGR Level 0 and RSYNCMGR Level 5 are the minimal manager levels required to support the sync point resync command to perform resynchronization.

### Sync Point Overview

Figure 3-88 (on page 946) shows the message flow of a source server connecting to a target server, accessing and updating an RDB, and then committing the transaction and terminating the conversation.

A conversation to the target server is created and the EXCSAT command is used to exchange server attributes. In this example SECMGR Level 5 is used and the commands ACCSEC and SECCHK create a verified end-user name at the target system. SYNCPTMGR does not in itself require the use of SECMGR. The ACCRDB command creates a conversation to a target RDB.

After creating an RDB conversation, the source and target SYNCPTMGRs must exchange log information and unit of work ID. A SYNCCTL (sync point control) command is used to request log information from the target SYNCPTMGR which is returned in a SYNCLOG (sync point log) object. Next a unit of work ID is sent to the target SYNCPTMGR using a SYNCCTL command. A send *uowid* SYNCCTL command does not have an expected reply.

The EXCSQLIMM command is an example of a command that updates the target RDB.

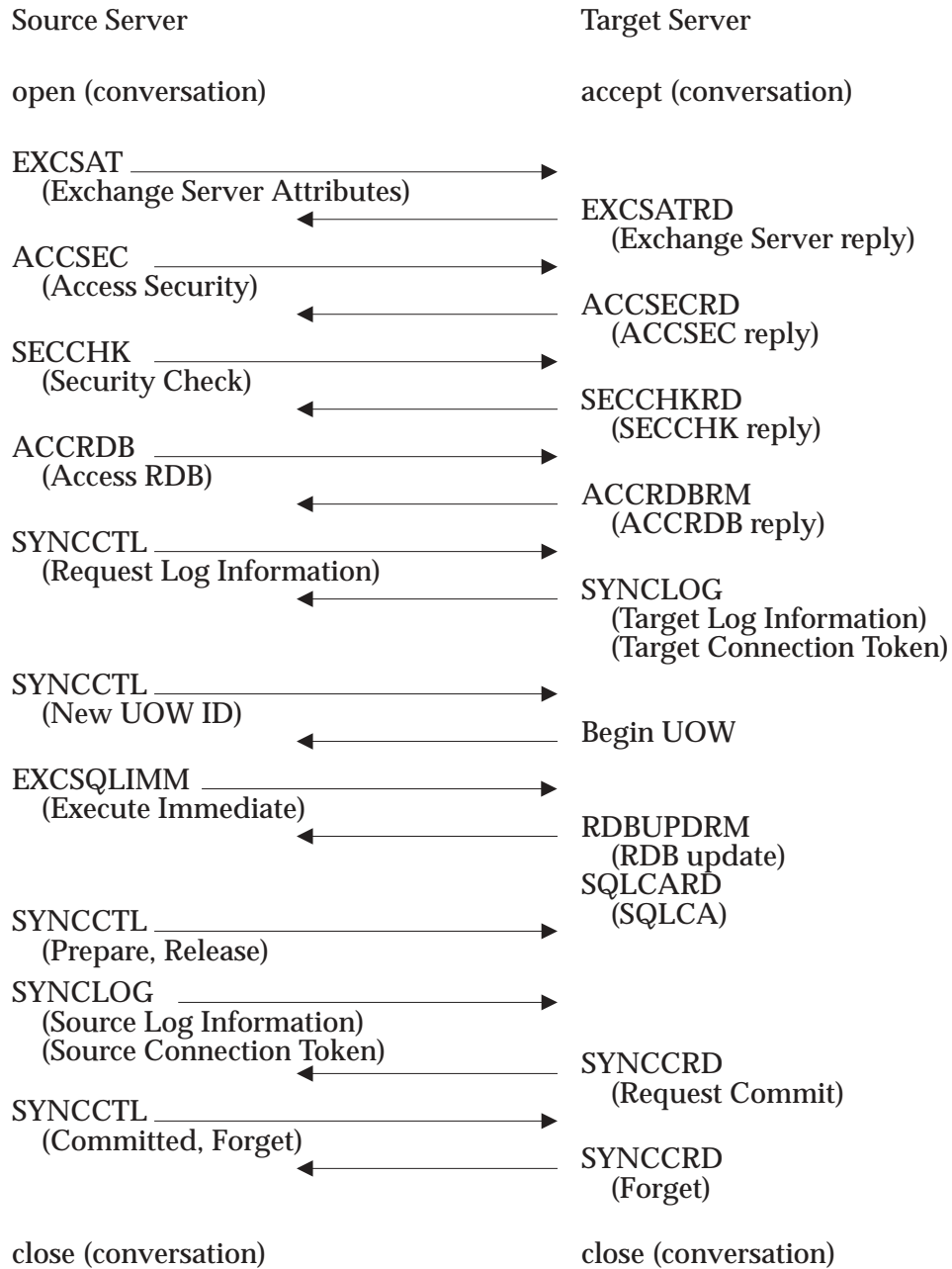
To commit the unit of work, a SYNCCTL command specifying prepare is sent to the target SYNCPTMGR along with log information for the source SYNCPTMGR in a SYNCLOG command data object. In the example, the target SYNCPTMGR responds with SYNCCRD (sync point control reply data) set to request commit. The source SYNCPTMGR completes the two-phase commit by sending SYNCCTL set to committed. This is acknowledged by a SYNCCRD reply set to forget. Since release was specified along with the prepare, both servers then terminate the conversation.

It is important to understand from the figure that:

- SYNCLOG is used by the SYNCPTMGRs to exchange log information.

An additional instance variable, CNNTKN, which is unrelated to the log information, is used to uniquely identify an instance of a server when multiple instances of a server have the same unit of work identifier (UOWID). This token is used to associate a resynchronization request to a specific server instance.

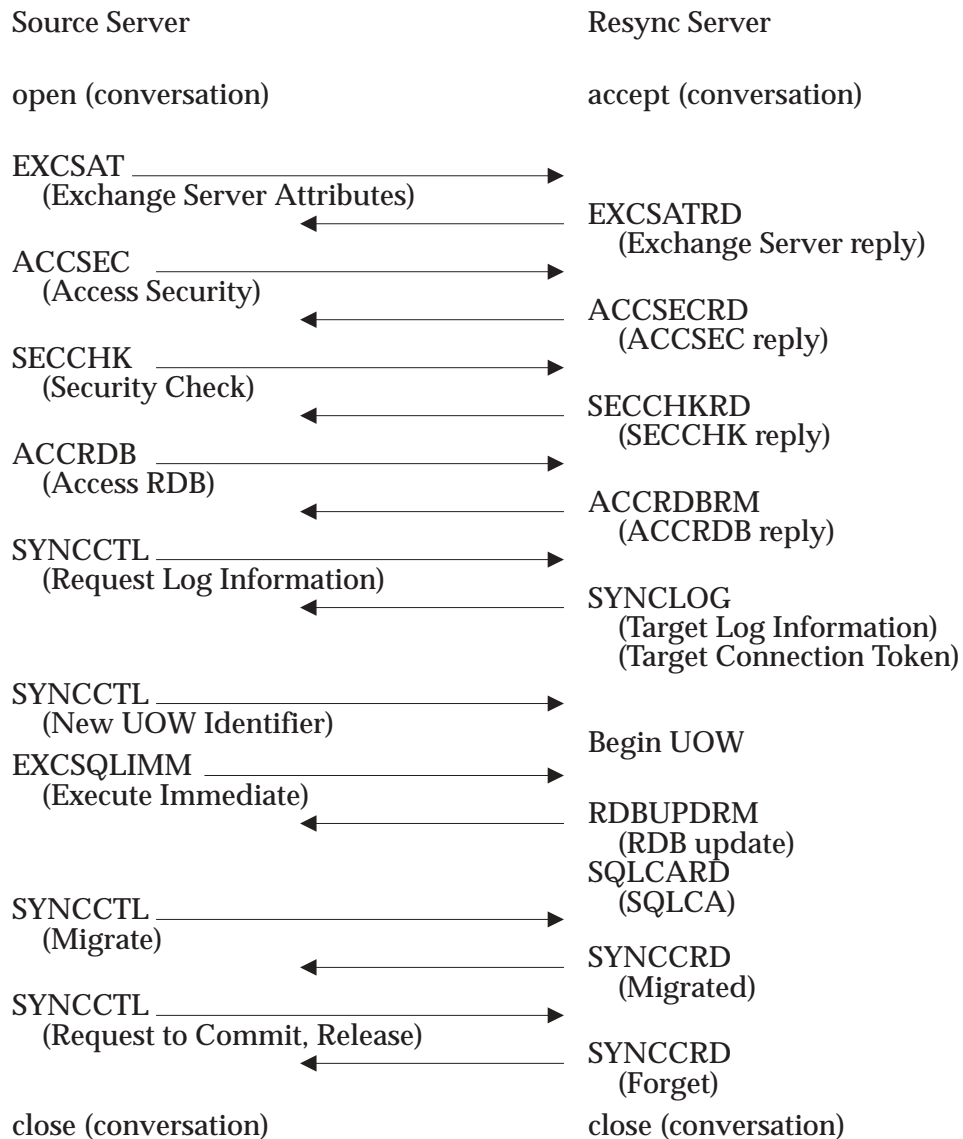
- A SYNCCTL command is used to send a unit of work ID to the target SYNCPTMGR at the beginning of each transaction. The unit of work ID along with the information in the SYNCLOG would be required for resynchronization if failures occur during the commit process.
- SYNCCTL commands and SYNCCRD replies are used to implement the two-phase commit protocol.
- Conversations which use the services of SYNCPTMGR can only be terminated normally by specifying release during a sync point operation.



**Figure 3-88** Illustration of Sync Point Flow

The above figure shows the source server committing changes in the RDB at the target server acting as the resync server sync point log.





**Figure 3-89** Illustration of Resync Server Sync Point Flow

The above figure shows the source server committing changes in the RDB at the target server acting as the resync server.

**Sync Point Log Object**

The purpose of this object is to provide log and resynchronization information to the source and target SYNCPTMGRs. This information is used to perform resynchronization if a system or communication failure occurs during the second phase of commit processing. The object contains the following information:

- Log name
- Log timestamp

- *rdbname*
- Network address used for resynchronization
- Connection token

SYNCLOG information is required to be provided by each SYNCPTMGR. A target SYNCPTMGR at Level 5 provides a SYNCLOG object as a reply to a SYNCCTL command that specifies request log information. A SYNCLOG object of the source SYNCPTMGR is sent to each target SYNCPTMGR participating in the unit of work on the SYNCCTL prepare command.

When the source SYNCPTMGR does not have a log, the resync server's SYNCLOG is used. This causes the target SYNCPTMGR to perform resynchronization with the resync server SYNCPTMGR instead of the original source SYNCPTMGR.

A list of SYNCLOG objects is sent to the resync server on the SYNCCTL request to commit command. The list contains a SYNCLOG from each of the participant SYNCPTMGRs that replied request to commit using SYNCCRD reply message.

Log names should be unique. It is recommended that the log name be generated by concatenating a unique network name like the domain host name with the resynchronization network address.

Logs should not be deleted while resync work remains to be completed. To identify this situation, a log timestamp is required. The timestamp is generated at the time the log is created. If a log name is received with a log timestamp having a different time value than the previous log timestamp, the log is considered cold or new. In all other cases, the log is considered warm. If two SYNCPTMGRs use mismatching logs or resync to the wrong SYNCPTMGR, data inconsistencies can occur since the log is different than what was being used during commit processing. Hueristic damage due to a cold log is detected during resynchronization.

The *rdbname* parameter in a SYNCLOG for a target server should match the *rdbname* that was specified in the ACCRDB for that server. In the case of a source server with a log, the source server must supply a local *rdbname* for the source server SYNCLOG object. This *rdbname* should be associated with the log being used by the source SYNCPTMGR and may not necessarily be associated with any local RDB at the source server.

The network address to perform resynchronization may be different than the network address used to connect to the target server. The SYNCLOG object allows the SYNCPTMGR to specify a network address to be used when connecting to perform resynchronization. For TCP/IP conversations, a resync port address is specified. Optionally, the domain host name may be provided.

The connection token specifies a value that uniquely identifies a server instance. It is possible that a server may have multiple instances with the same unit of work identifier and the connection token is used to uniquely identify the specific instance. The concatenation of a local connection token and the partner connection token uniquely identifies the specific server instance of the unit of work identifier. The connection token is not to be consider part of the log information.

SYNCPTMGR log names, resynchronization network addresses, and connection tokens are required to be logged by each SYNCPTMGR with a log. The log name and resynchronization network address represent an instance of the SYNCPTMGR. The connection token represents a specific server instance managed by the SYNCPTMGR and is logged during commit processing. After a system failure, the log is read to determine the units of work that require resynchronization, the log name used at the time of the commit, where to connect to perform resynchronization, and the connection token associated with the unit of work.

### Sync Point Control Command and Reply

When an application issues a commit or rollback, sync point control messages are exchanged between the source SYNCPTMGR and all target SYNCPTMGRs to perform the two-phase commit. The source SYNCPTMGR is the coordinator of the two-phase commit. The target SYNCPTMGR is the participant SYNCPTMGR of the two-phase commit. The sync point control message consists of the following types:

- Sync Point Control Command
  - Prepare: solicits request commits (to participant)
  - Request to commit: solicits committed (to resync server)
  - Committed: unit of work is committed (to participant)
  - Forget: unit of work is complete (to resync server)
  - Forget: to release conversation not in current unit of work (to participant)
  - Rollback: unit of work is backed out (to participant)
  - Migrate: migrate resynchronization responsibilities (to resync server)
  - New uowid: send new unit of work identifier (to participant or resync)
- Sync Point Control Reply:
  - Committed: unit of work is committed (from resync server)
  - Migrated: resynchronization responsibilities migrated (from resync server)
  - Request to commit: solicits committed (from participant)
  - Rollback: unit of work is backed out (from participant or resync server)
  - Forget: unit of work is complete (from participant or resync server)

A conversation participates in a commit or rollback if a SYNCCTL new unit of work command was sent for the current unit of work.

To terminate a conversation, a commit must be performed successfully with the *release* parameter set. After the commit, the conversation is terminated by each SYNCPTMGR.

### Unit of Work Identifier

The unit of work identifier (UOWID) is maintained by the source SYNCPTMGR. The SYNCCTL command with the parm *uowid* set to the value of the unit of work identifier can be chained with the next DDM command after a successful commit or rollback to inform the target SYNCPTMGR of the new UOWID. When a new UOWID is received by a target SYNCPTMGR, the SYNCPTMGR is part of the current unit of work and participates in any commit or rollback. A target SYNCPTMGR cannot participate in a commit or rollback without knowing the unit of work identifier for the transaction.

**Source SYNCPTMGR With Log**

When an application decides to commit a unit of work, the source SYNCPTMGR is invoked to manage the two-phase commit and is called the coordinator. The coordinator initiates the first phase of the commit protocol by sending, serially or in parallel, the SYNCCTL prepare command to all of the target SYNCPTMGRs involved in the current unit of work. The target SYNCPTMGRs are called the participants of the two-phase commit. The SYNCCTL prepare command is used by the coordinator to determine whether the participants are willing to commit the unit of work. If the conversation is to be released by a participant after the commit is complete, the release flag is set in the SYNCCTL command.

Each participant that is willing to let the unit of work be committed, and which has updated recoverable resources, first force-writes a prepare log record and then sends a SYNCCRD request to commit reply to the coordinator and waits for the final decision (commit/rollback) from the coordinator. The unit of work at the participant SYNCPTMGR is now in the prepared state (also known as the in-doubt state).

If a participant does not have any updated recoverable resources or held cursors, it sends the SYNCCRD forget reply and releases locks and forgets about the unit of work. A read-only participant writes no log records. As far as this participant is concerned, it does not matter whether the unit of work ultimately gets rolled back or committed. So the participant, who is now known by the coordinator to be read-only, does not need to be sent a commit or rollback message by the coordinator.

Each participant that wants to have the unit of work backed out writes a non-forced rollback log record and sends a SYNCCRD rollback reply to the coordinator. Since a rollback vote acts like a veto, the participant knows that the unit of work will definitely be backed out by the coordinator. Hence, the participant does not need to get any more information from the coordinator. Therefore, the participant backs out the unit of work, releases its locks, and forgets about the unit of work.

After the coordinator receives the votes from all its participants, it initiates the second phase of the protocol. If all the votes were either request to commit vote and forget votes, then the coordinator moves to the committing state, force writes a commit log record, and sends the SYNCCTL committed command to all participants. If any participant voted forget, the participant is not sent the committed message. If all participants voted forget, and the coordinator also has not updated any recoverable resources, there is no second phase of the protocol and no log records are written by the coordinator. If any participant voted rollback, then the coordinator moves to the rollback state, and writes a non-forced rollback log record, and sends the SYNCCTL rollback command to all participants who did not vote rollback or forget and then forgets the unit of work.

Each participant, after receiving a committed message moves to the committing state, force-writes a commit log record, sends a SYNCCRD forget reply to the coordinator, and then commits the unit of work and forgets it. Each participant, after receiving a rollback message moves to the rollback state, writes a non-forced rollback log record and rolls back the unit of work. The unit of work is forgotten.

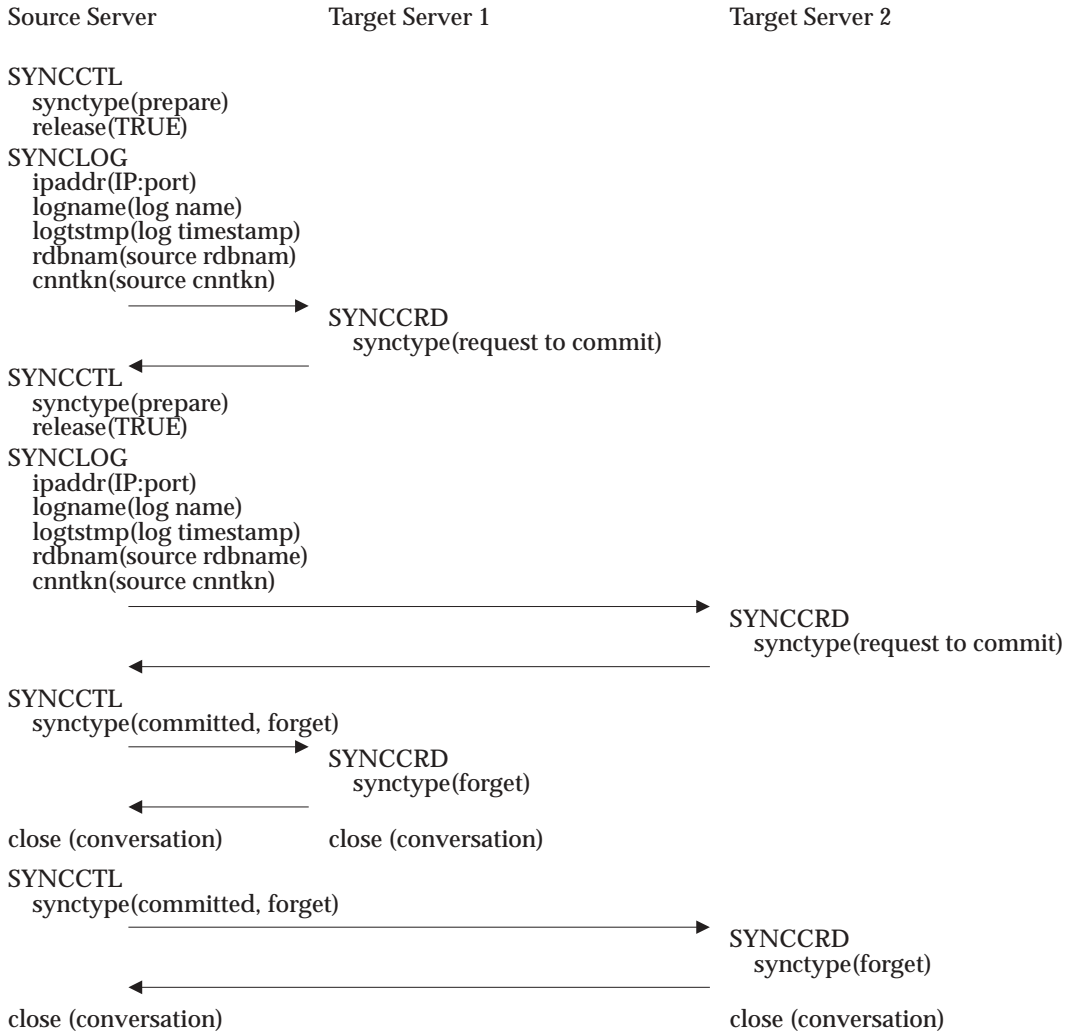
When the coordinator is in the committing state the coordinator waits to receive responses from all the participants that were sent a message in the second phase, writes a forget record and forgets the unit of work.

If the release flag was set TRUE, the conversation is closed but only if the commit was successful (no rollback votes). In order to meet this requirement, any server that is released must go through both phases of commit, even if it is a read-only server. That is, when being released, a server must return a SYNCCRD request to commit reply, and can only honor the release when it

receives the committed decision.

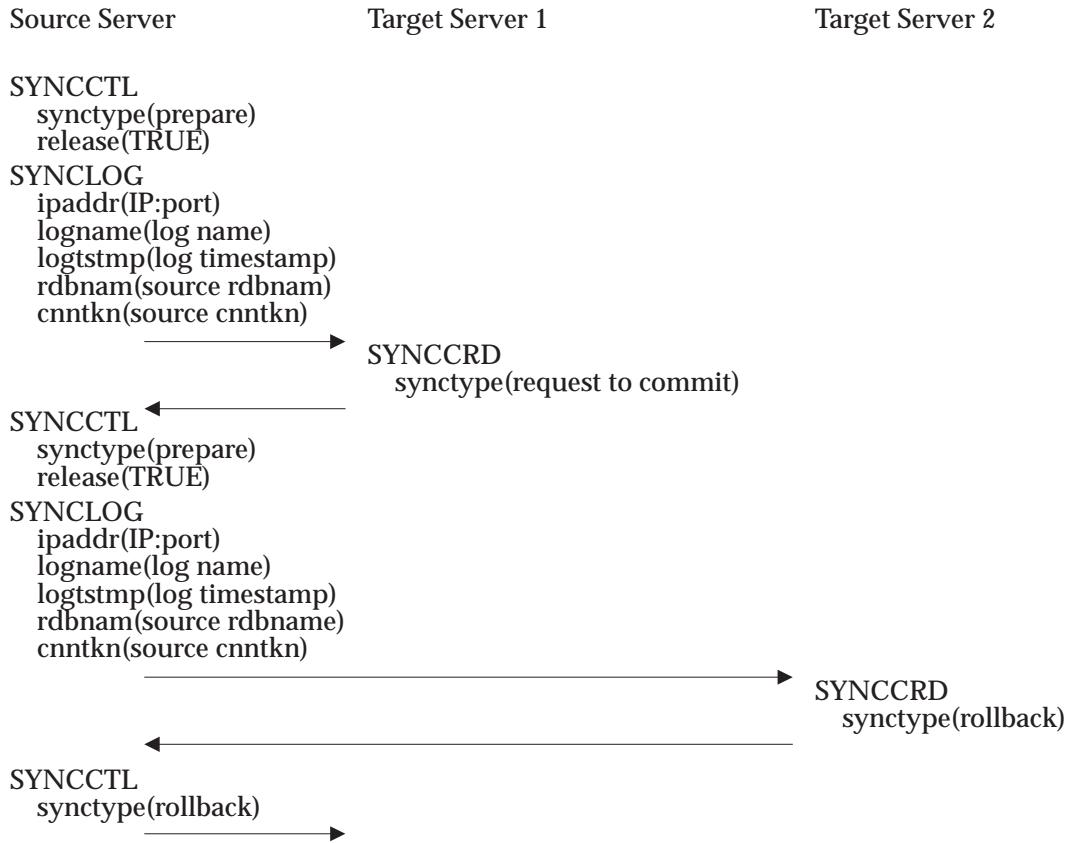
When an application decides to rollback a unit of work, the source SYNCPTMGR writes a non-forced rollback record and sends the SYNCCTL rollback command to all participants. The participant server writes a non-forced rollback log record and forgets the unit of work. If the participant is read only, no log records have to be written.

The following figures show the commit and rollback flows for a source SYNCPTMGR with a log.



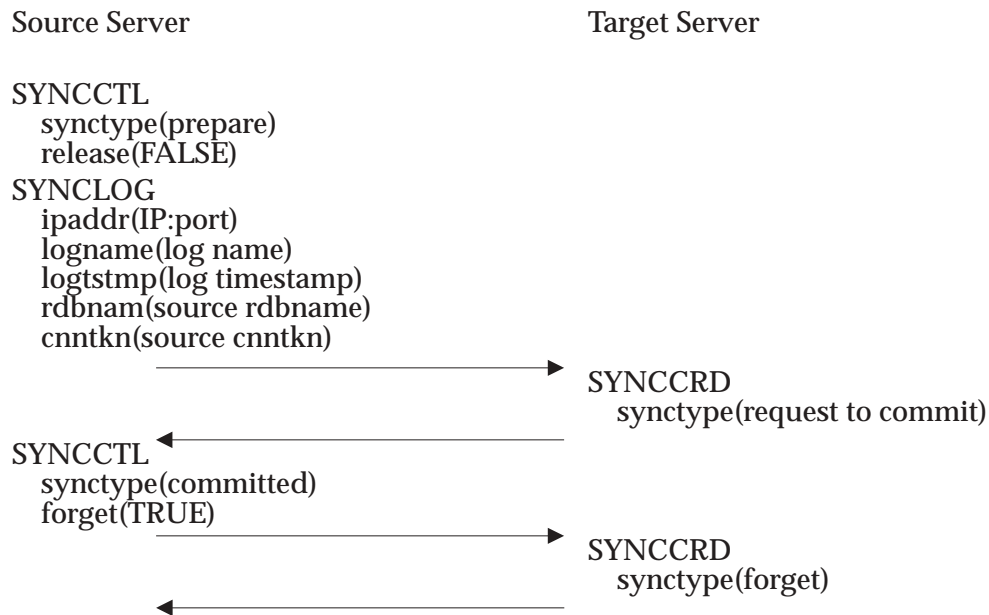
**Figure 3-90** Multiple Released Connections Commit Flow

The above figure illustrates the four sync point control message exchange between a source server with a log and two target servers. The four message exchange is performed when the connections are marked for release at the beginning of the commit. The prepare message may be sent to the servers in parallel. The SYNCCTL committed command requires an explicit SYNCCTRL forget reply in this example because the connections are being terminated at part of the sync point operation.



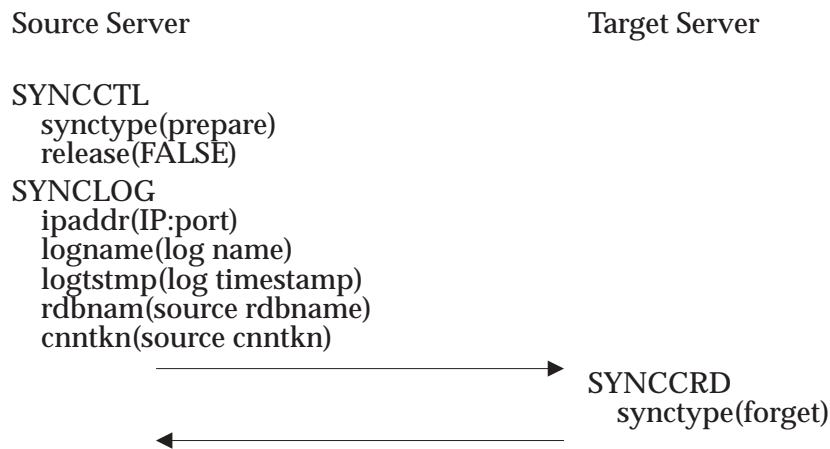
**Figure 3-91** Multiple Released Connections Rollback Flow

The following flows show scenarios involving only a single target server. The source SYNCPTMGR with a log always maintains the final commit decision on a unit of work. This is done for availability and requires that a full two-phase commit logic is used even in the case of a single target server.



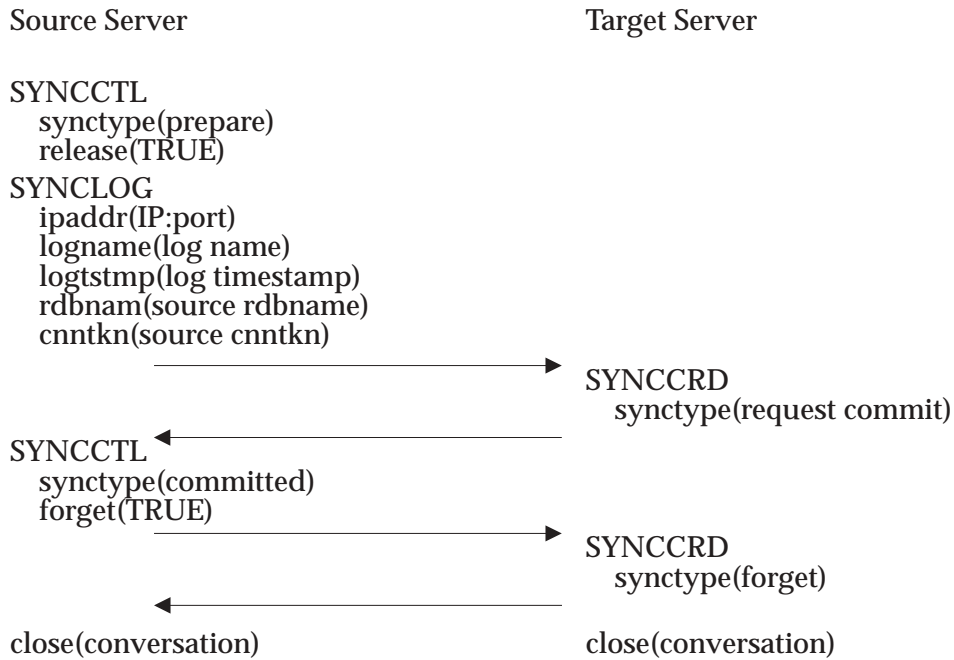
**Figure 3-92** Commit With Updates Flow

The above flow illustrates the four sync point control messages that are exchanged when a unit of work has updated resources at the target server.



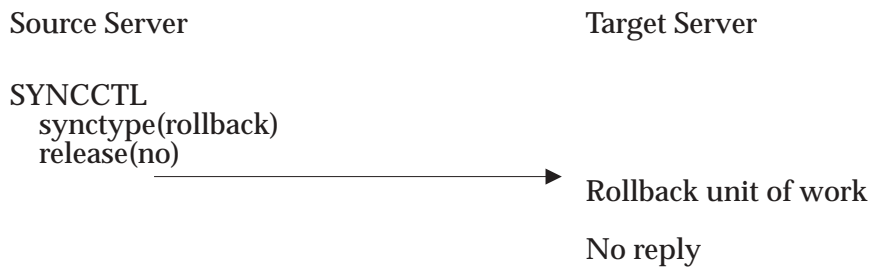
**Figure 3-93** Commit Without Updates Flow

The above flow illustrates the two sync point control messages exchanged when the server is read-only.



**Figure 3-94** Released Connection Commit With/Without Updates

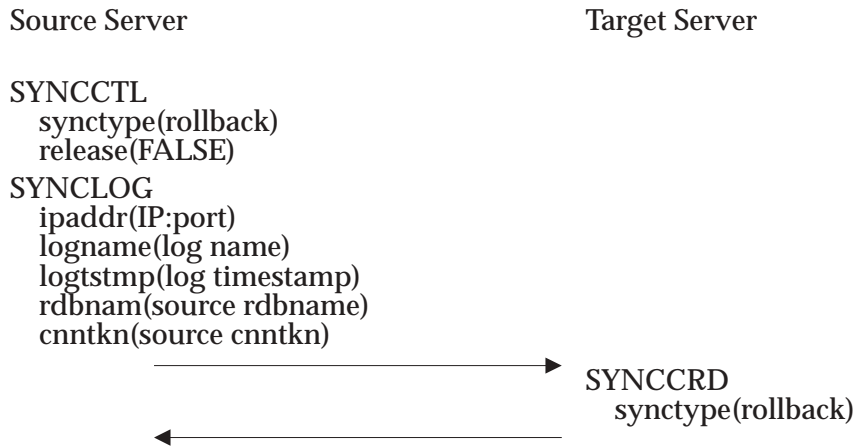
The above flow illustrates the four sync point control messages exchanged when the connection is released after the commit is complete. The sync point forget message is sent before closing the connection.



**Figure 3-95** Source SYNCPTMGR Initiated Rollback

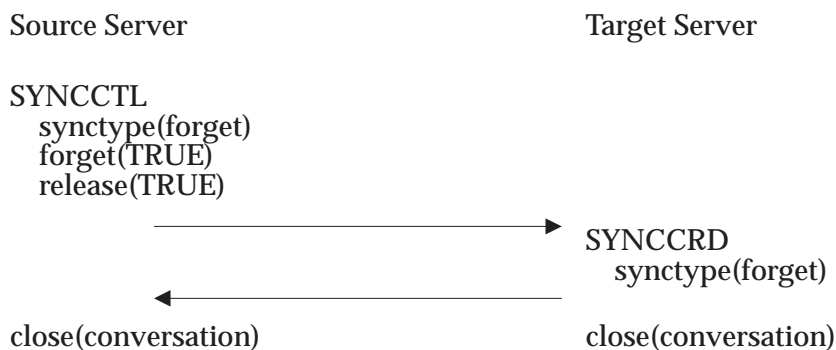
The above flow illustrates the one sync point control message when a source server with a log initiates a rollback.





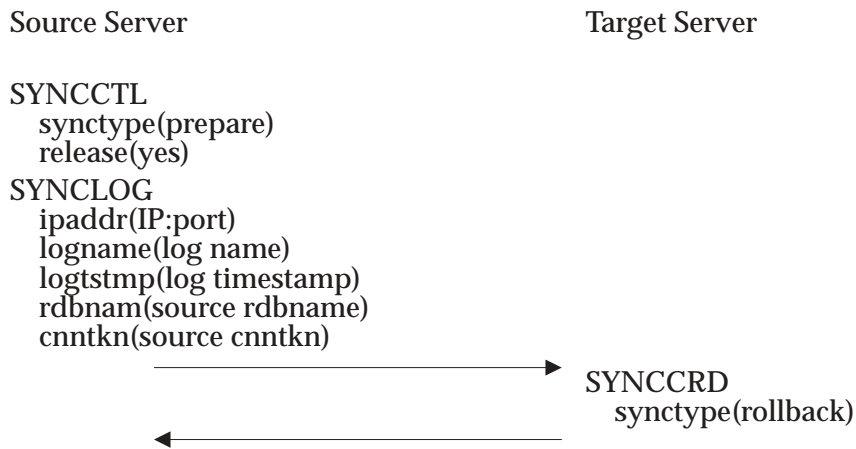
**Figure 3-96** Target SYNCPTMGR Initiated Rollback Flow

The above flow illustrates the one sync point control messages when a target server is not in the current unit of work. The sync point new unit of work command must not have been sent since the last sync point operation.



**Figure 3-97** Source SYNCPTMGR Terminates a Conversation

The above flow illustrates the two sync point message exchange when a source server with a log terminates a conversation for a target server not part of the current unit of work, but the source server may have outstanding implied forgets. Otherwise, the source server can close the connection if the connection is not part of the current unit of work and the source server has no outstanding implied forgets.



**Figure 3-98** Target SYNCPTMGR Initiated Rollback Flow

The above flow illustrates the two sync point control messages exchanged when a source server with a log rolls back the unit of work. Release(yes) was specified on the prepare, but not performed because the server responded with rollback.

**Implied Forget**

SYNCPTMGR Level 5 utilizes an optimization called implied forget that is used between a participant and the coordinator when the conversation is not to be released after completing the commit. The implied forget optimization is such that the forget response to the committed SYNCCTL message is deferred. In fact, it is implied via the response to the next DDM request sent to the participant. This optimization saves a message. The coordinator sends the committed message with no forget and does not wait for a response. The participant receives and processes the message but does not generate a response. It waits for the next DDM request (see Figure 3-99 (on page 957)).

If the next request is to close the connection, and the SYNCCTL UOWID command has not been sent to start the next UOW, the coordinator can send a SYNCCTL forget message (release=TRUE, forget=TRUE) to the participant in which case the participant will return a forget reply and close the connection. The forget reply implies the forget for the prior unit of work (see Figure 3-97 (on page 955)).

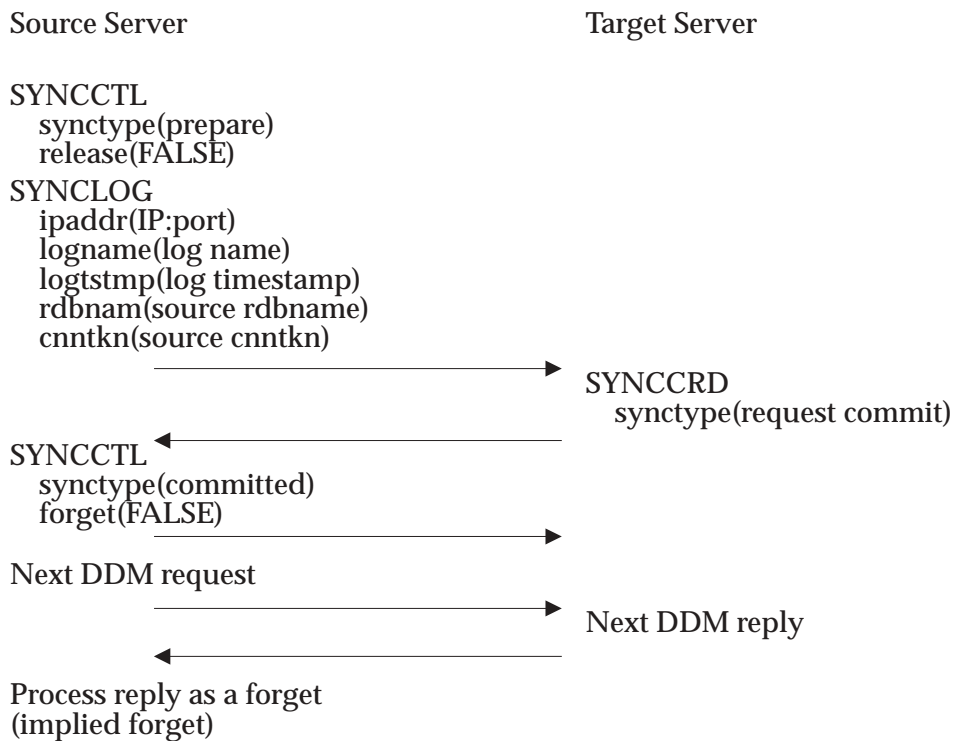
The source SYNCPTMGR without a log must maintain knowledge of a unit of work until all implied forgets and forget sync point control messages are received from each participant. As soon as all forget messages are received, the forget SYNCCTL command is sent to the resync server while processing the next commit.

If the target server returns the following error reply messages, the target server has detected an error condition. These responses do not imply a forget and if a previous unit of work is expecting an implied forget the unit of work may be in-doubt. Resynchronization is required with the target SYNCPTMGR.

Error responses that do not imply a forget are:

- ABNUOWRM    Abnormal End Unit Of Work Condition (see *ABNUOWRM* (on page 39))
- AGNPRMRM    Permanent Agent Error (see *AGNPRMRM* (on page 65))
- CMDCHKRM    Command Check (see *CMDCHKRM* (on page 173))

PRCCNVRM      Conversational Protocol Error (see *PRCCNVRM* (on page 625))  
 RSCLMTRM      Resource Limits Reached (see *RSCLMTRM* (on page 778))  
 SYNTAXRM      Data Stream Syntax Error (see *SYNTAXRM* (on page 989))



**Figure 3-99** Commit Using an Implied Forget Flow

The above flow illustrates the three sync point control messages that are exchanged when a unit of work has updated resources at the target server. Since the conversation is not marked for release, implied forget processing is used to acknowledge that commit has completed.

**Source SYNCPTMGR Without a Log**

When the source SYNCPTMGR acting as the coordinator does not have a log and decides to commit a unit of work, it migrates resync responsibilities to a target SYNCPTMGR called the resync server. To migrate, the migrate SYNCCTL command is sent to a target SYNCPTMGR participating in the current unit of work. The resync server acknowledges the migrate request by sending the SYNCCRD reply with migrated specified. The target SYNCPTMGR is the resync server for the current unit of work.

The coordinator initiates the first phase of the commit protocol by sending, serially or in parallel, the SYNCCTL prepare command to all target SYNCPTMGRs acting as participants except the resync server to determine whether they are willing to commit the unit of work. After the coordinator receives the votes from all the participants, it sends the outcome of the first phase to the resync server.

- If at least one participant voted request to commit and no participants voted rollback, then the coordinator sends the SYNCCTL request to commit command to the resync server. The message includes a list of SYNCLOG objects received from the participants that voted request to commit. The resync server force-writes a commit log record, and acknowledges by

sending the SYNC CRD committed reply to the coordinator, and commits. The coordinator is in the committing state. The coordinator sends each of the other participants the SYNC CTL committed command.

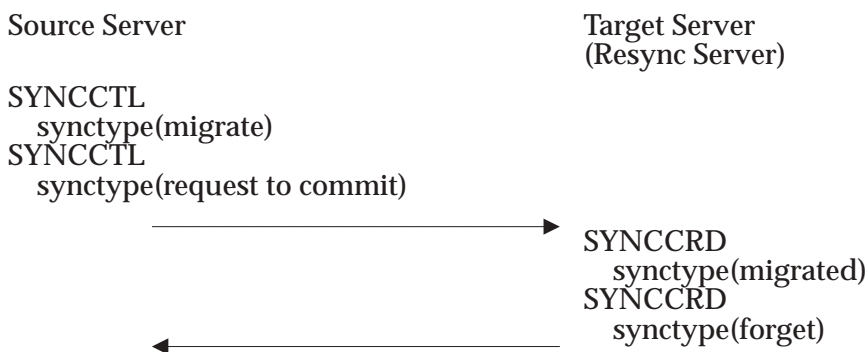
- If all participants voted "forget", the coordinator sends the SYNC CTL request to commit message with an empty participant list to the resync server. There is no second phase of the protocol. The resync server writes a "forget" record and sends the SYNC CRD forget reply message to the coordinator to complete the unit of work.
- If a participant voted rollback, the coordinator sends the SYNC CTL rollback command to the resync server. The resync server writes a non-forced rollback log record, and sends the forget sync point control response to the coordinator. The coordinator moves to the rollback state and sends the SYNC CTL rollback command to all participants who did not vote rollback or forget. The resync server is not sent a SYNC CTL rollback command.

Each participant after receiving a committed message moves to the committing state, force-writes a commit log record, sends a sync point control forget response to the coordinator, and then commits the unit of work and forgets it. Each participant, after receiving a rollback message moves to the rollback state, writes a non-forced rollback log record, sends a forget response (if requested) to the coordinator, and backs out the unit of work. The unit of work is forgotten.

The conversation is closed if the release flag was set TRUE and if there were no rollback votes. As in the case of the source SYNCPTMGR with log, a server that replies forget in the first phase of commit must not close the conversation until it receives the close from the source server.

The coordinator, after receiving the responses from all the participants that were sent a message in the second phase, sends a SYNC CTL forget command to the resync server. The resync server writes a forget record and forgets the unit of work.

If there is only one target server involved in the unit of work, then that server by default becomes the resync server. In this case, the source SYNCPTMGR can optimize the message flow by chaining together into one transmission the SYNC CTL migrate and SYNC CTL request to commit commands. If there are multiple target servers this chaining is not allowed.



**Figure 3-100** Commit Flows SYNCPTMGR Without Log

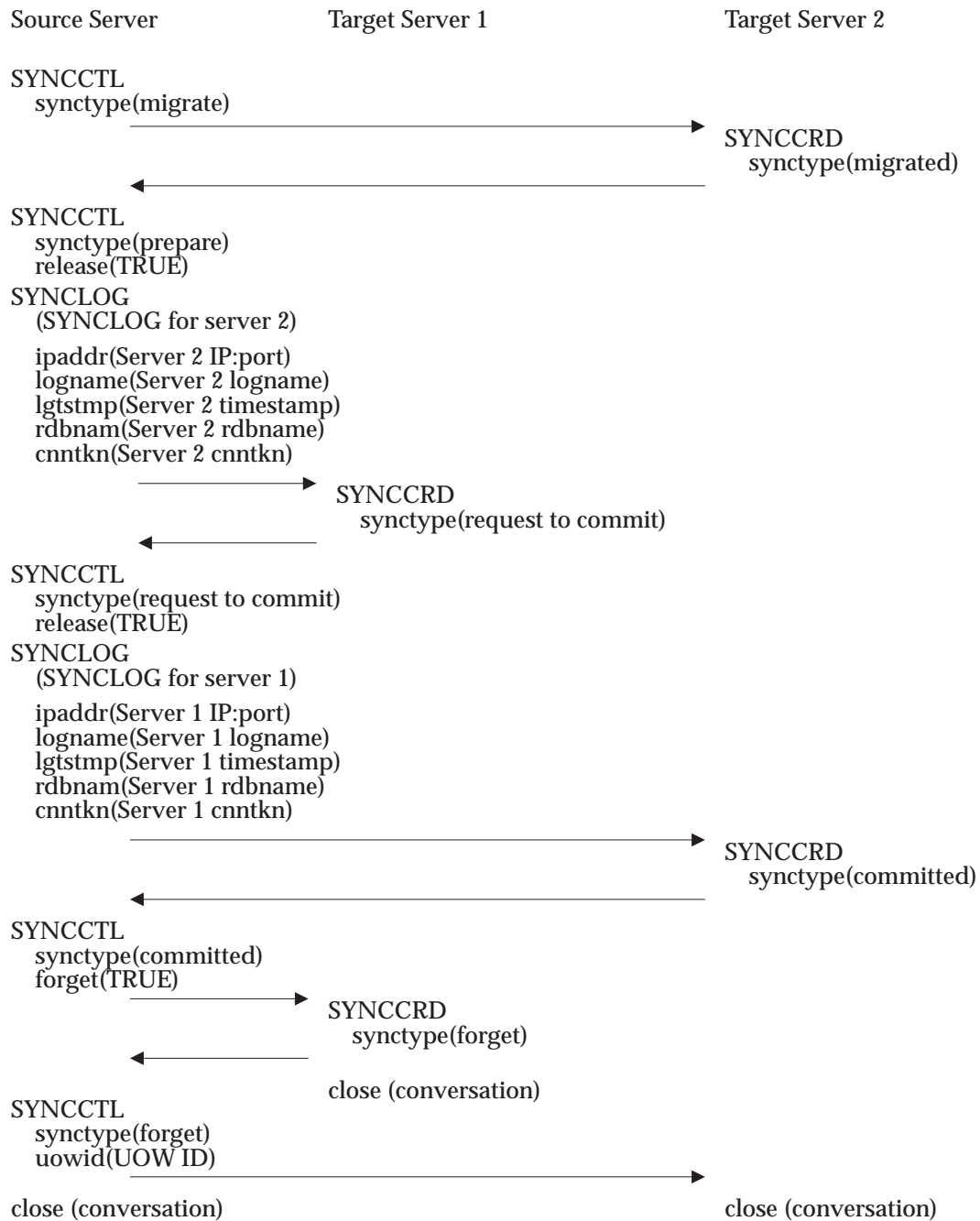
The above figures illustrates the commit protocol when the source SYNCPTMGR does not have a log. The target server is the only target server involved in the transaction and by default is the resync server.



**Figure 3-101** Rollback for SYNCPTMGR Without Log

The above figure illustrates the sync point control message to perform a rollback.

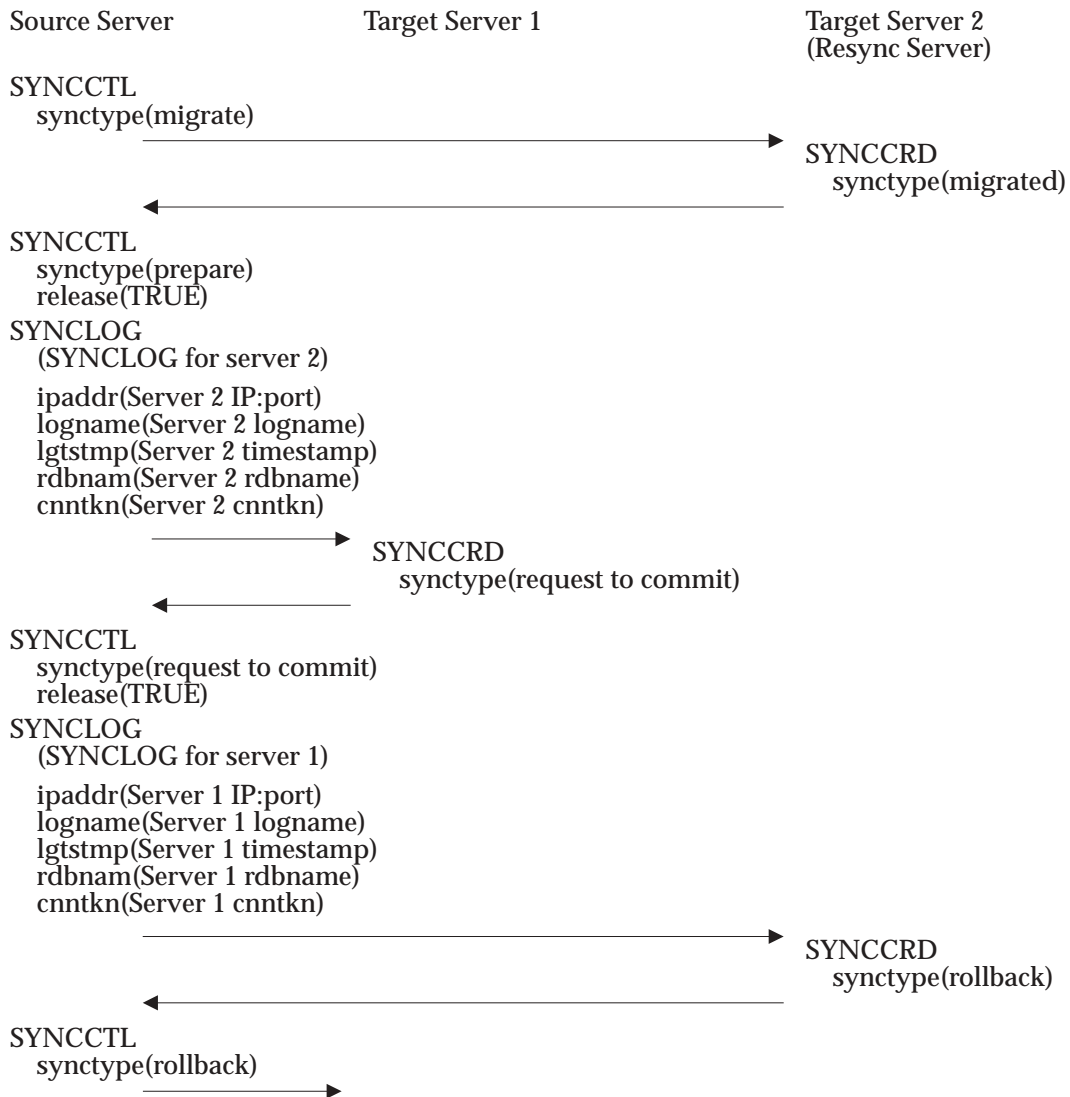
The following figures show the sync point control flows for source SYNCPTMGR with two target SYNCPTMGRs.



**Figure 3-102** Multiple Released Connections Commit Flow

The above figure illustrates the commit processing between a source server without a log and two target servers where server 2 is designated as the resync server. The SYNCCTL request to commit command contains a list of the participant’s SYNCLOG objects that voted to commit. The list allows the resync server to know the possible SYNCPTMGRs that may attempt to resync. Since the outcome of the unit of work is commit, the conversation is terminated.

The following figure shows the flows when a target server votes rollback for a source SYNCPTMGR without a log.



**Figure 3-103** Multiple Released Connections Rollback Flow

The above figure illustrates the commit processing between a source server without a log and two target servers where server 2 is designated as the resync server. The migrate SYNCCTL command contains a list of the participant’s SYNCLOG objects. The list allows the resync server to know the possible SYNCPTMGRs that may attempt to resync if the commit fails. Even though release(yes) was specified on the prepare and request to commit commands, since one of the servers voted rollback, the close conversation is not performed.

**Considerations for Sync Point Commands with No Replies**

Sync point commands that do not have replies are:

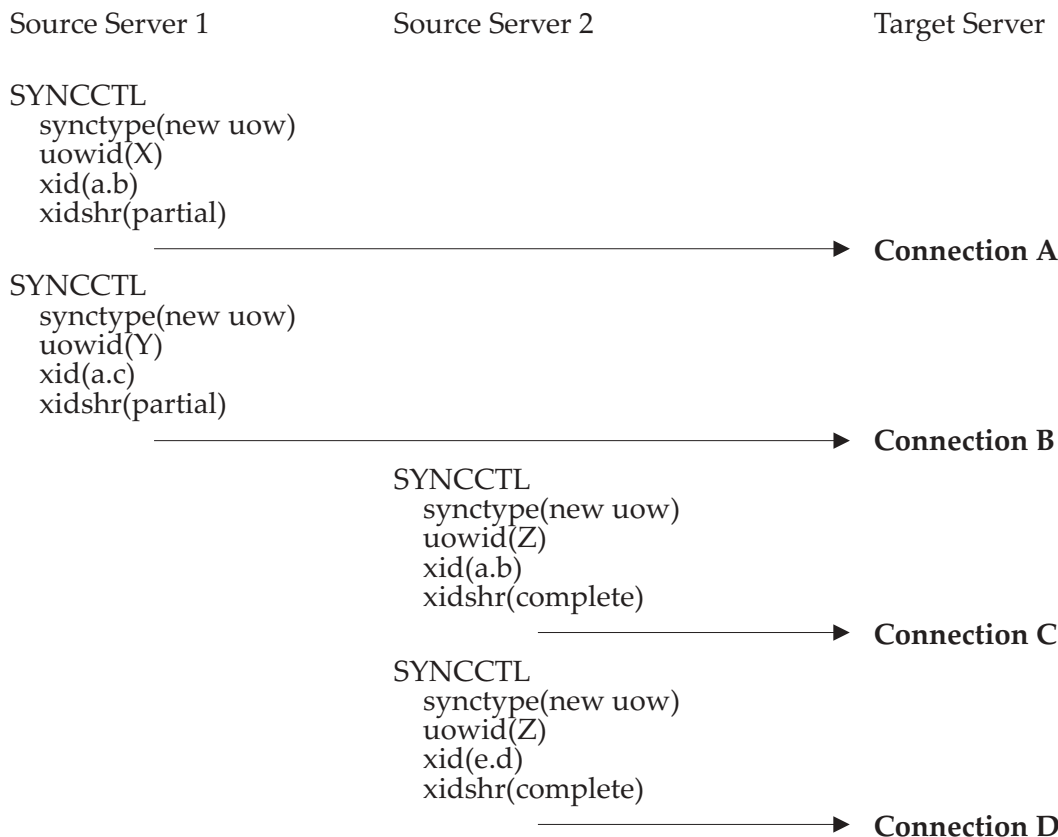
- SYNCCTL synctype(new unit work ID)
- SYNCCTL synctype(forget)
- SYNCCTL synctype(committed) if forget set to FALSE
- SYNCCTL synctype(rollback)

These commands must be sent with a DSSFMT dsstype of X'5'.

For CMNAPPC conversations, to minimize the overall elapsed time for a sync point operation, a source server should use the SNA flush option when sending sync point commands without replies in cases where an indefinite amount of time may occur between the next DDM command. The target server should specify FILL=LL to ensure that it is notified as soon as the command is received rather than for a full buffer or for permission\_to\_send to be received.

**Resource Sharing**

The figure below shows multiple protected connections at a target server, with various XIDs.



**Figure 3-104** Set of Connections that can Share Resources

In the above example, Connection A has enabled partial sharing, therefore any connections attempting to use or access resources and locks that are being held by A will result in a deadlock, except connections whose XID value matches exactly with that of A. In the sample above, only



connection C meets this requirement, and hence will be able to share A's held resources without resulting in a deadlock.

Connection B has enabled partial sharing also, but no other connection has the same *XID* as B. Therefore all connections will result in a deadlock if they try to access any resources or locks that are being held by B.

Connection C has enabled complete sharing; that is, any connection whose *gtrid* value of the *XID* matches can share resources. Both A and B's *gtrid* value matches C's *gtrid* value. Therefore both A and B can share resources and locks held by C, but D cannot.

Connection D has enabled complete sharing but no other connection's *gtrid* value matches D's *gtrid* value. Therefore none of the connections can share D's held resources and locks.

### Sync Point Resync Command and Reply

If a network or system outage interrupts the commit operation, the commit decision may not be known by a participant *SYNCP TMGR*. When the outcome is unknown, the unit of work is in-doubt. The coordinator or resync server always knows the outcome of the unit of work. To resolve the in-doubt, the participant *SYNCP TMGR* sends the sync point *SYNCRSY* command to the coordinator or to the resync server. The coordinator or resync server replies with the *SYNCRRD* reply that indicates the outcome of the unit of work. Enough unit of work state and log information is exchanged so that both sides can determine the appropriate action to take and resolve the in-doubt unit of work. When the resync command is complete, the unit of work is completed and forgotten. The sync point resync message identifies five possible commit states:

- **In-doubt:** outcome of transaction is in-doubt because of interrupted sync point operation.
- **Unknown:** commit state is unknown because the transaction is still in progress. (Specified on the resync sync point reply when the resync server has not received the outcome of the unit of work from the coordinator (race condition).)
- **Commit:** unit of work committed. (Specified on the resync sync point command or reply by the coordinator or resync server when the coordinator has committed the unit of work.)
- **Reset:** unit of work is rolled back or forgotten. (Specified on the resync sync point command or reply by the coordinator or resync server when the coordinator has rolled back the unit of work or by a participant who has forgotten the unit of work.)
- **Cold:** indicates that a cold start was performed and the log needed for recovery is no longer accessible.

The two-phase commit protocol requires the coordinator or resync server to remember units of work that are committed until all participant *SYNCP TMGRs* who may be in-doubt have been informed of the commit decision. Thus, the coordinator or resync server must include in the commit log record the *SYNCL OG* information of all participants who voted request to commit. This information could be in one or more non-forced log records written before the forced commit record. The participant must include in the prepare log record the *SYNCL OG* information of the coordinator. This information could be in a non-forced log record written before the forced prepare record. To resolve in-doubt units of work, *SYNCRSY* messages have to flow between the *SYNCP TMGRs*.

Either the coordinator or the resync server is responsible for initiating resynchronization (in-doubt resolution) when the commit log record is forced to the recovery log. The responsibility is ended when all participants which voted request to commit to the prepare message have acknowledged the coordinator's committed message with a forget reply message. This is indicated on the recovery log by the existence of a forget log record. The participant's responsibility begins when the prepare log record is forced to the recovery log. The

responsibility is ended when forget is sent as the response to the committed request. This is indicated on the log by the existence of forget log record.

As part of recovery from a system failure, the SYNCPTMGR reads the recovery log and accumulates in volatile storage information relating to units of work that were executing the commit protocol at the time of the failure. It is this information in volatile storage that is normally used to:

- Answer queries from other locations who were participants of units of work coordinated by the recovering location
- Send unsolicited information to other locations which were participants in migrated units of work coordinated by the recovering location
- Send a query to the location coordinating a unit of work which is in-doubt because of the failure

When a SYNCPTMGR manager at a participant location reconstructs, from the recovery log, the state of units of work, and finds that a unit of work is in the prepared state, it periodically tries to contact the coordinator or resync server to find out how the unit of work should be resolved. When the coordinator or resync server informs the participant of the outcome of the unit of work, the participant writes a commit or rollback log record and completes the unit of work.

If the SYNCPTMGR at a coordinator or resync server finds a unit of work in the committing state, it periodically tries to send the commit decision to all participants that have not yet acknowledged the receipt of the message. After acknowledgments are received from all participants, the coordinator or resync server writes the forget log record and forgets it. When a coordinator or resync server receives a resync inquiry message from a prepared participant, it looks at its information in virtual storage. If it has information which says that the unit of work is in the committing state, then it sends the SYNCRRD committed response. If the coordinator can find no information about the unit of work, the reset response is returned to indicate the unit of work was rolledback.

In addition to the list of units of work needing resolution which is accumulated by reading the recovery log after a system failure, a SYNCPTMGR may have to add units of work to the list during normal operation. This occurs when a communication failure occurs between a coordinating and a participating location.

- If the coordinator notices the failure of a participant (loss of communication) while waiting for the latter to send its vote to the prepare message, then the coordinator backs out the unit of work. If the failure occurs while the coordinator is waiting for a response to the committed message, then the coordinator adds the unit of work to the list of units of work needing resolution with participants.
- If a participant notices the failure of the coordinator before the former had voted request to commit and is in the prepared state, then it backs out the unit of work. If the failure occurs after the participant has gone into the prepared state, then the participant adds the unit of work to the list of unit of work needing resolution with the coordinator.

All failures are ultimately recovered. However, an administrator may choose to force the resolution of one or more in-doubt units of work prior to the recovery process. This is a human decision at a participant location and can lead to inconsistent recovery. This is often referred to as heuristic decision and heuristic damage. Resynchronization is required if the SYNCPTMGR sent a sync point control committed message but a network outage occurred before the forget could be received or if the SYNCPTMGR sent a request to commit sync point control message but a network outage occurred before the committed or sync point control rollback message could be received. In these cases, the unit of work is in-doubt and resynchronization must be

performed.

The resync server may not know if a participant is in-doubt. This occurs when the sync point forget message is received by the coordinator and not the resync server. When the coordinator has received all the sync point forget messages, the sync point forget message is sent to the resync server. If the forget is never received by the resync server, resynchronization must be performed between each participant and the resync server. After resynchronization is complete, the unit of work is forgotten.

To perform resynchronization the source server (not necessarily a source server in the original transaction) initiates a conversation to a target server using the resync address logged in the SYNCLOG object. The two servers exchange server attributes using the EXCSAT command. The source server can send one or more SYNCRSY commands (one for each outstanding unit of work at the target RDB). Finally the source server sends a SYNCRSY end command to terminate the conversation.

Figure 3-105 (on page 966) illustrates a source SYNCPTMGR which was a participant server in the original transaction initiating resync for two in-doubt units of work with a target SYNCPTMGR which was a coordinator in the original transaction. The first resync request is for an in-doubt unit of work that has committed. The second request is for an in-doubt unit of work that has rolled back.

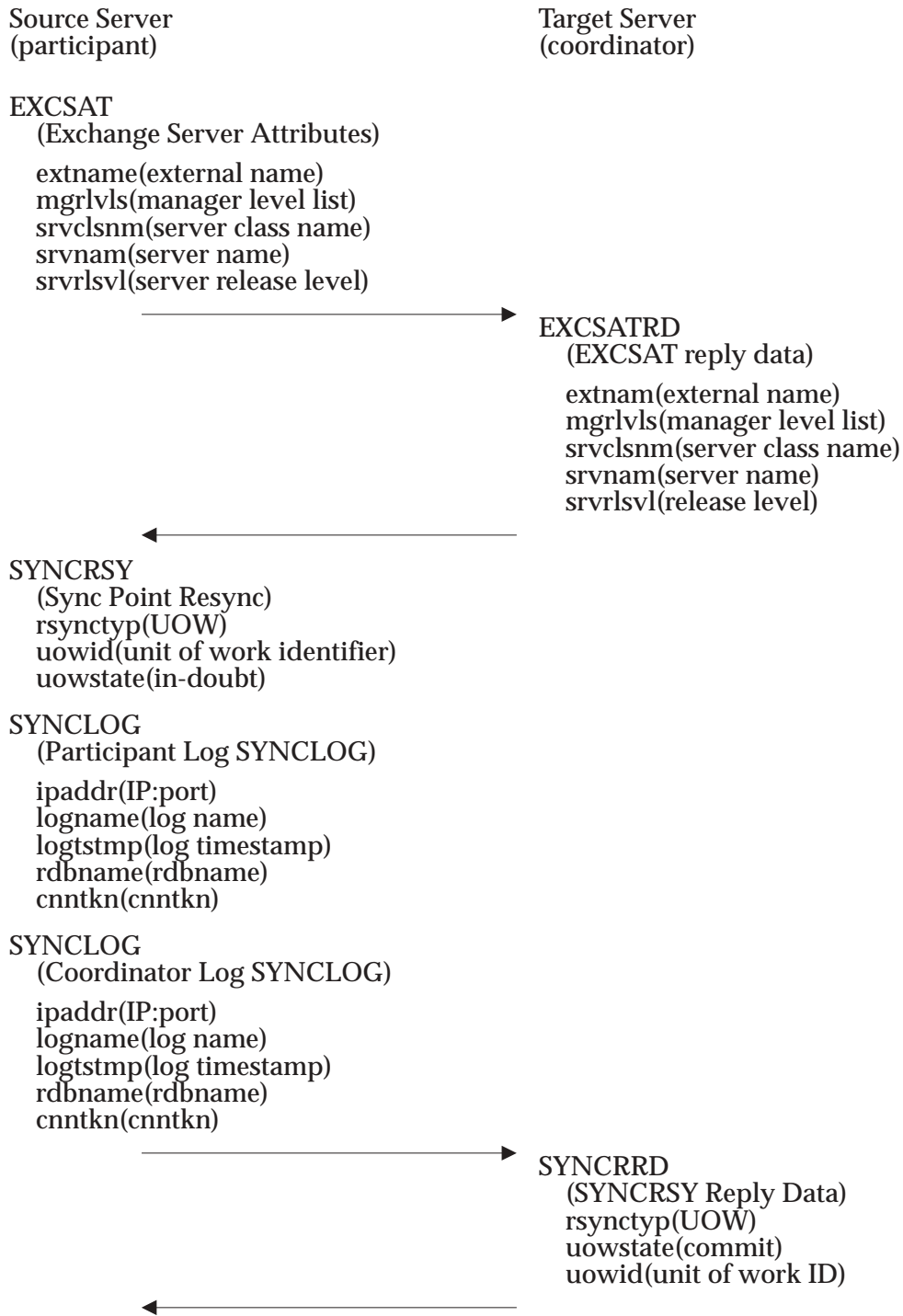
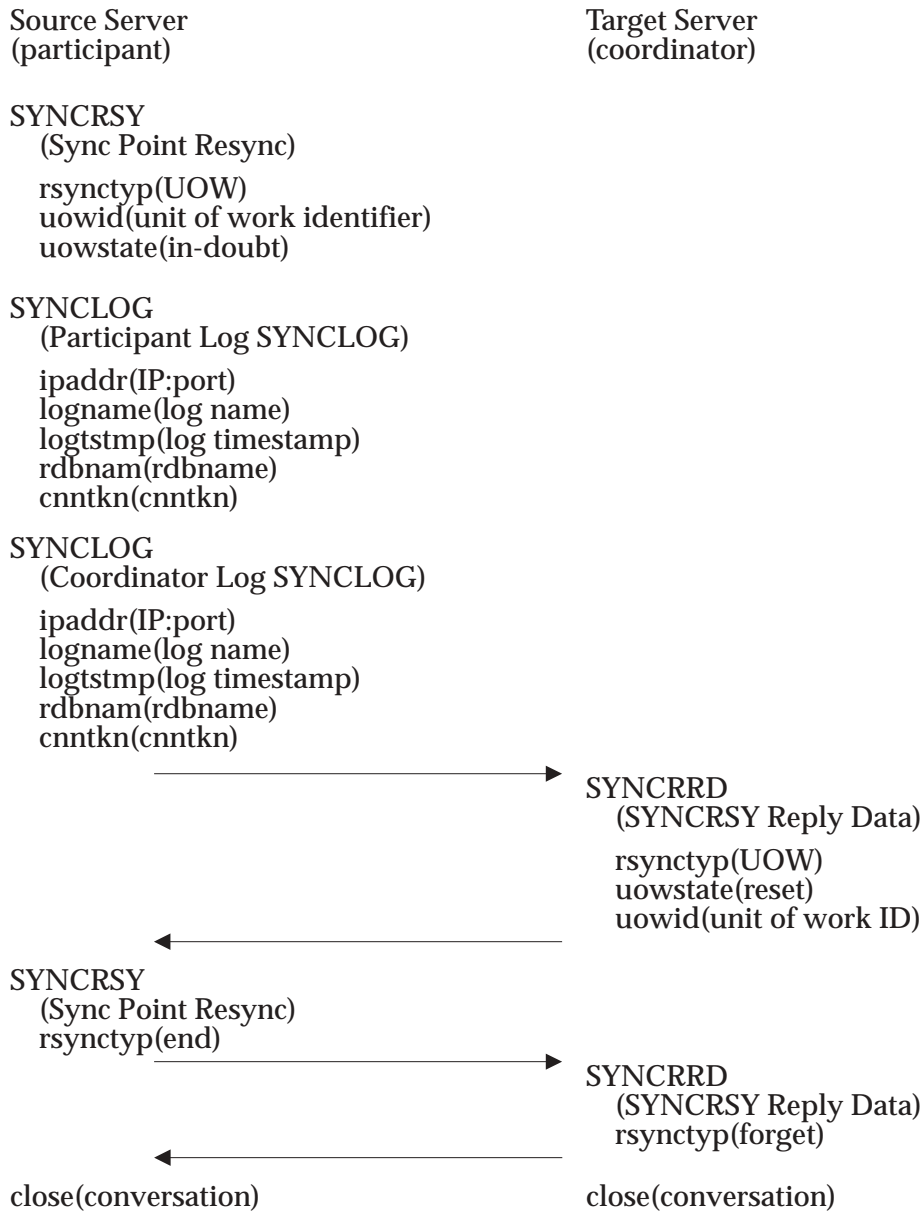


Figure 3-105 Resync Connection (Part 1)



**Figure 3-106** Resync Connection (Part 2)

**Cold Start Considerations**

A cold start may occur at one or more of the SYNCPTMGRs associated with an interrupted transaction. This may make access to the log required for recovery impossible. If a SYNCPTMGR cannot access the log indicated by the SYNCLOG record because of a cold start, it replies to the resync request with uowstate set to COLD.

**Race Condition**

A resynchronization race condition may occur if a participant cannot send the SYNCCRD request to commit reply. In this case, the participant is in-doubt and is required to perform resynchronization with the resync server. The race condition occurs when the resync server tries to process the sync point SYNCRSY command before receiving the request to commit or SYNCCTL rollback command from the coordinator. The resync server may decide to immediately rollback the unit of work, wait for the coordinator to send the SYNCCTL request to the commit command, or reply with the commit state of unknown. If the resync server returns with a commit state unknown, the participant must retry until the resync server replies with the commit decision.

If a communication error occurs during the processing of the SYNCCTL request to the commit command between the coordinator and the resync server, the commit decision is not known by the coordinator. In this case, the coordinator has to return a non-zero sqlstate identifying that the commit decision is unknown.

Figure 3-107 (on page 969) illustrates the race condition when a participant tries to resync before the resync server knows the outcome of the unit of work. In this example, the resync server waits for the commit decision before replying to the resync request.

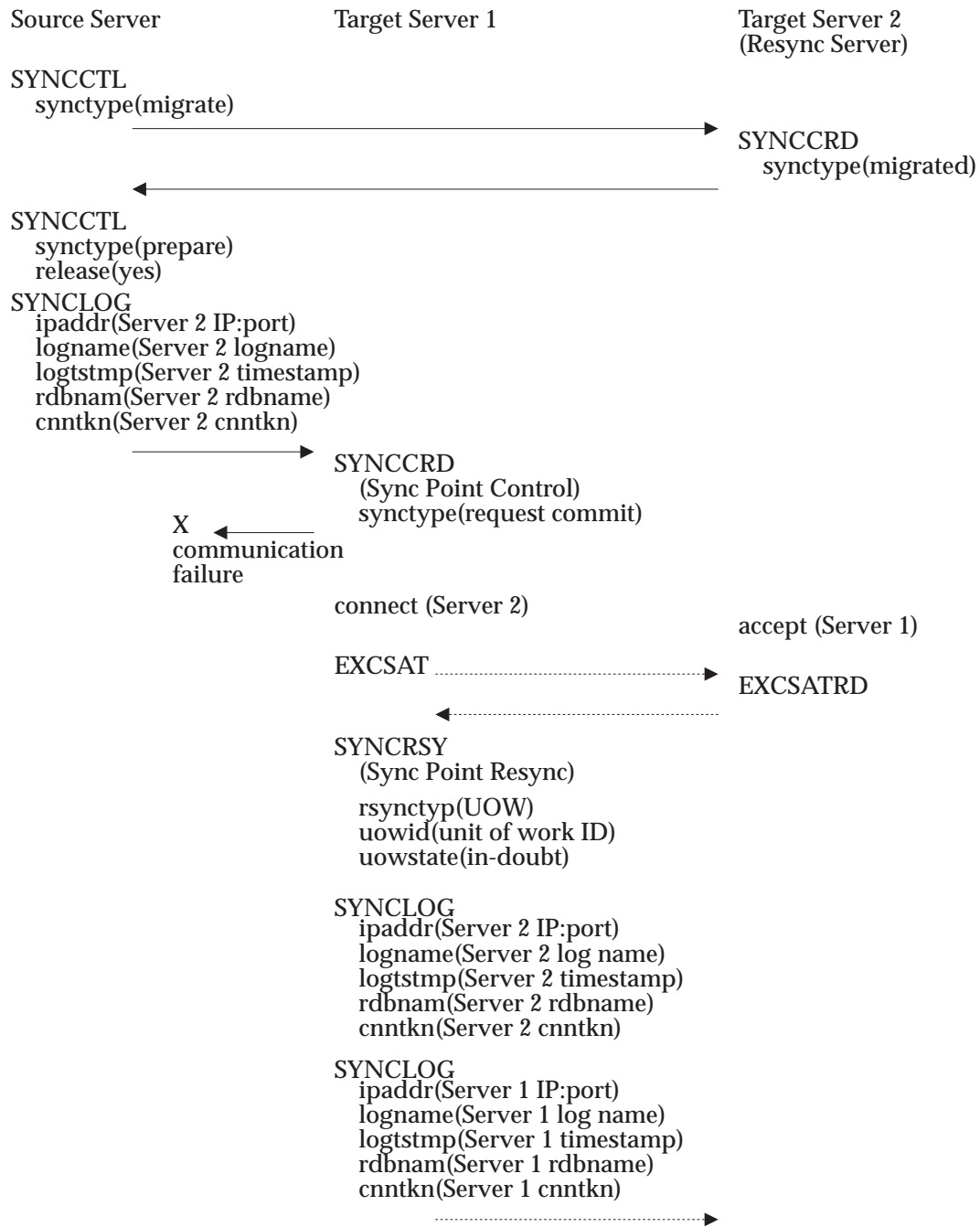
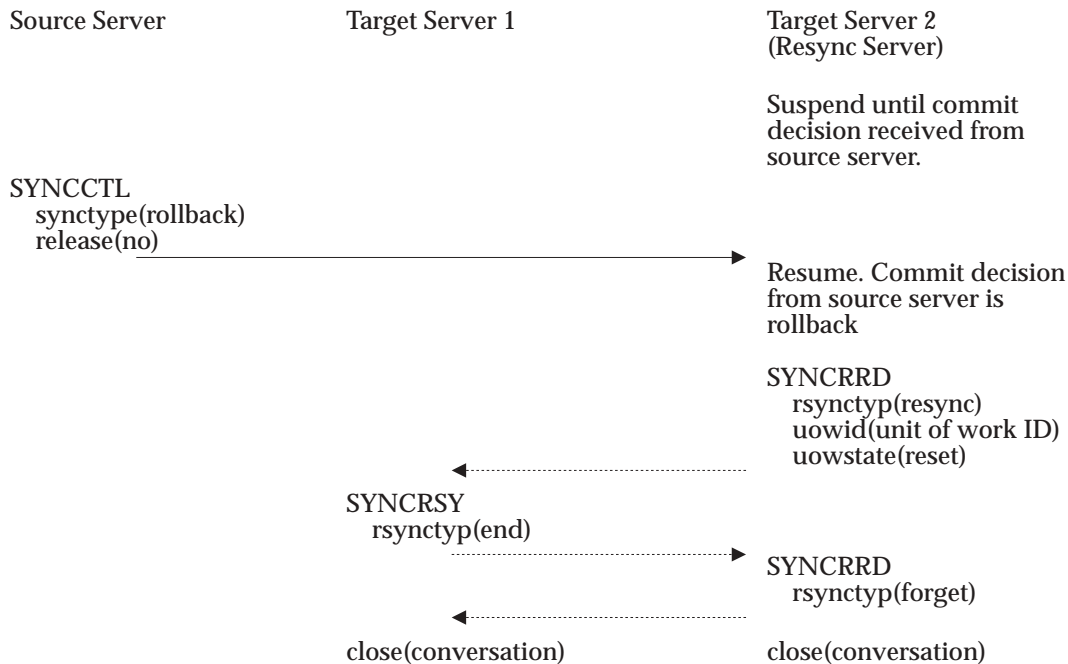


Figure 3-107 Multiple Connections Race Condition (Part 1)



**Figure 3-108** Multiple Connections Race Condition (Part 2)

**Resynchronization with Multiple Servers**

Parallel databases support the concept of a group of servers with access to the same RDB. Each server in this group has its own network address. If one of the servers involved in a transaction is a parallel server, the problem can arise that at resynchronization time the server has been restarted on an alternate processor with a network address different from the address on the SYNCLOG object that was exchanged at the time of the transaction. Resynchronization is only possible by connecting to the server instance which has the necessary log information concerning the unit of work.

To find the current address of a server instance, it is required that the TCP port number or the SNA transaction program name be part of the resynchronization network address contained in the SYNCLOG object. Furthermore, the TCP port number or SNA transaction program name must uniquely identify the server instance within the server group and cannot change even if the server is restarted on an alternate processor and has a new IP address or SNA luname.

To initiate resync with a server that is a member of server group, the server initiating resync should first attempt to open a conversation using the resync address from the SYNCLOG object exchanged during the transaction.

If this fails because the target server does not respond, then it is possible that the target server has been restarted using a different address than the address recorded in the SYNCLOG object. In this case, the source server should initiate resync by establishing a conversation using an IP address obtained using the domain host name provided on the SYNCLOG object.



### Log Records

In order to maintain protected resources in consistent states even if a failure occurs during commit processing, the SYNCPTMGR logs unit of work state information in non-volatile storage. The logged state information represents the point reached in the sync point operation, the characteristics of the protected resources, and the role of the local SYNCPTMGR (source or target). From the logged information, the SYNCPTMGR can determine the right actions to take to return the resources to a consistent state.

There are two ways to write information to non-volatile storage: forced write and non-forced write. A forced write operation does not complete until the information is written to storage. A non-forced write completes when the information is put into log buffers. Forced write operations take longer to complete, but the information is guaranteed to be available following a system failure.

Sync point state tables provided below specify when log records are to be written and when the unit of work is in-doubt. If a unit of work is in-doubt and the commit operation fails, the SYNCPTMGR is required to start resync protocols to resolve the in-doubt. After a system failure occurs, the log is read, the state of in-doubt units of work are recreated, and resync is initiated.

The information that must be logged by a SYNCPTMGR consists of the following:

- uowid: unit of work ID
- uowstate: current state of unit of work
- SYNCLOG object(s)

A participant server logs the SYNCLOG object of the resync or coordinator server. The coordinator server that has a log writes the SYNCLOG objects of all participating update servers. A resync server logs the SYNCLOG objects of all participating update servers.

### Sync Point Manager Tables

Two protocol state tables define the sync point control operation, one for the source SYNCPTMGR (coordinator) and one for the target SYNCPTMGR (participant or resync server). The two tables specify the interrelationship between the current state, the incoming events that occur, and the resulting outgoing actions and state.

### Conventions

In the SYNCPTMGR state table, the intersection of an incoming event (row) and a state (column) forms a cell. A non-blank cell represents a combination of an incoming event and a state that is defined for the SYNCPTMGR. A non-blank cell consists of one or more sub-cells. A sub-cell consists of the following:

- Zero, one, or more predicate values
- Zero, one, or more outgoing actions
- A resultant state

Prerequisite predicate value(s)

-Outgoing action(s)

(Resultant state)

**Figure 3-109** Sub-Cell Contents

A logical NOT (!) before a predicate indicates the logical negation of the predicate. The logical AND (&) between predicates indicate that both predicates must be true. The logical OR (|) between predicates indicate that only one predicate must be true.

**Table 3-18** Coordinator SYNCPTMGR With Log

Events		States				
		reset	prepare	rollback	committed	resync
E1	Application issues commit.	(prepare: E4)				
E2	Application issues rollback.	(rollback: E6)				
E3	Application ends and connection must be terminated.	-IFO -CLOSE (reset: E1) IFO (reset: E9)				
E4	Send prepare and the coordinator log information to each participant in the current unit of work. Set release if connection is to be closed after commit.		ACF -ABORT (rollback: E6) -ACF (prepare: E5)			
E5	Receive reply to prepare from each participant.		ACF   RRB -ABORT (rollback: E6) -ACF&RRC& -RRB -Write CLR -COMMIT (committed: E7) -ACF&AFT -COMMIT (reset: E1)			
E6	Send rollback to each participant.			(reset: E1)		
E7	Send committed to each participant that voted request to commit to prepare.				ACF (resync: E11) -ACF (committed: E8)	
E8	Receive forget or an implied forget from participants that voted request to				ACF (resync: E11) -ACF&REL -CLOSE -Write FLR	

Events		States				
		reset	prepare	rollback	committed	resync
	commit to prepare.				(reset: E1) -ACF&-REL -Write FLR (reset: E1)	
E9	Send forget to participant. Set release and forget.	ACF (resync: E11) -ACF (reset: E10)				
E10	Receive reply to forget.	ACF (resync: E11) -ACF -CLOSE -Write FLR (reset: E1)				
E11	Send resync to each in-doubt participant that voted request to commit to notify the participant of the outcome of the unit of work.					ACF (resync: E11) -ACF&AFT -Write FLR (reset: E1)
E12	Receive resync from a participant that voted request to commit to determine outcome of the unit of work.					ACF   -AFT (resync: E11) -ACF&AFT -Write FLR (reset: E1)

**Legend**

- ABORT      Rollback unit of work.
- ACF        Any conversation failure.
- AFT        Received all forget or performed resync with each participant.
- CLOSE     Close any released connections.
- CLR        Force write committed log record.
- IFO        Implied forget outstanding.
- COMMIT    Commit unit of work.
- FLR        Non-forced forget log record.
- REL        A connection is released.
- RRB        Receive at least one rollback in response to prepare.
- RRC        Receive at least one request to commit in response to prepare.

Events		States				
		reset	prepared	forget	committed	resync
E1	Receive prepare from coordinator.	RR&CMT -Write PLR (prepared: E5) -RR&CMT&-REL -COMMIT (forget: E6) -RR&CMT&REL -COMMIT (prepare: E5) REL&RBK -ABORT (reset: E8) -REL&RBK -ABORT (reset: E1)				
E2	Receive rollback from coordinator.	-ABORT (reset: E1)				
E3	Receive forget from coordinator.	(reset: E4)				
E4	Send forget to coordinator.	-CLOSE (reset: E1)				
E5	Send request to commit to coordinator.		CF (resync: E9) -CF (committed: E7)			
E6	Send forget to coordinator.			REL -CLOSE (reset: E1) -REL (reset: E1)		
E7	Receive commit decision from coordinator.				CF (resync: E9) -CF&REL&RCT -COMMIT -Write FLR (forget: E6) -CF&RCT&EF -COMMIT -Write FLR (forget: E6) -CF&-REL& RCT&-EF -COMMIT -Write FLR (reset: E1) -CF&RRB -ABORT	

Events		States				
		reset	prepared	forget	committed	resync
					-Write FLR (reset: E1)	
E8	Send rollback to coordinator	(reset: E1)				
E9	Send resync to coordinator to determine outcome of unit of work.					CF (resync: E9) -CF&CMT -COMMIT -Write FLR (reset: E1) -CF&RBK -ABORT -Write FLR (reset: E1)
E10	Receive resync from coordinator to provide outcome of the unit of work.					CF (resync: E9) -CF&CMT -COMMIT -Write FLR (reset: E1) -CF&RBK -ABORT -Write FLR (reset: E1)

**Legend**

- ABORT      Rollback unit of work.
- CF              Conversation failure.
- CLOSE      Close released connection.
- CLR            Forced committed log record.
- CMT           Unit of work state is committed.
- COMMIT      Commit unit of work.
- EF              Explicit forget requested.
- FLR            Non-forced forget log record.
- PLR            Force write a prepared log record.
- RBK            Unit of work state is reset (rollback).
- RCT            Receive committed from coordinator.
- REL            Connection released.
- RR              Recoverable resources and any open held cursors.
- RRB            Receive rollback from coordinator.

Table 3-19 Coordinator SYNCPTMGR With No Log

Events		States				
		reset	migrate	rollback	prepare	committed
E1	Application issues commit.	(migrate: E3)				
E2	Application issues rollback.	(rollback: E5)				
E3	Send migrate to a participant to act as the resync server for the current unit of work.		CF (rollback: E5) ¬CF (migrate: E4)			
E4	Receive migrated from resync server.		CF (rollback: E5) ¬CF (prepare: E6)			
E5	Send rollback to each participant or to each participant that voted request to commit.			(reset: E1)		
E6	Send prepare and the resync server log information to each participant in the current unit of work. Set release indicator in prepare if conversation is to be closed after commit.				ACF (rollback: E5) ¬ACF (prepare: E7)	
E7	Receive vote from each participant.				ACF (rollback:E5) ¬ACF&RRB (rollback: E5) ¬ACF&¬RRB (prepare: E8)	
E8	Send request to commit to resync server with log information from each participant that voted request to commit.				CF -RSYNC (reset: E1) ¬CF (prepare: E9)	

Events		States				
		reset	migrate	rollback	prepare	committed
E9	Receive reply from resync server.				CF -RSYNC (reset: E1) ¬CF&RFT&REL -CLOSE (reset: E1) ¬CF&RFT&¬REL (reset: E1) ¬CF&RRB (rollback: E5) ¬CF&RCT (committed: E10)	
E10	Send committed to each participant that voted request to commit.				ACF (reset: E1) ¬ACF (committed: E11)	
E11	Receive forget or an implied forget from participants that voted request to commit.					ACF (reset: E1) ¬ACF (committed: E12)
E12	Send forget to resync server					REL -CLOSE (reset: E1) ¬REL (reset: E1)

**Legend**

- ACF        Any conversation failure.
- CLOSE    Close any released connections.
- CF        Connection failure.
- REL       A connection is released.
- RRB       Receive rollback reply to request to commit or prepare.
- RCT       Receive committed reply to request to commit or prepare.
- RFT       Receive forget reply to request to commit.
- RSYNC    Close connections to force resynchronization with resync server.

Table 3-20 Resync Server SYNCPTMGR

Events		States				
		reset	prepare	rollback	committed	resync
E1	Receive migrate from coordinator.	CF -ABORT (reset: E1)  ¬CF (prepare: E2)				
E2	Send migrated to coordinator.		CF -ABORT (reset: E1)  ¬CF (prepare: E3)			
E3	Receive request from coordinator to determine outcome of unit of work.		CF -ABORT (reset: E1)  ¬CF&RRB -ABORT (reset: E1)  ¬CF&RRC&RBK -ABORT (rollback: E5)  ¬CF&RRC&CMT&NPT -Write CLR -COMMIT (prepare: E4)  ¬CF&CMT&¬NPT -Write CLR -COMMIT (committed: E6)			
E4	Send forget to coordinator to complete the unit of work.		REL -CLOSE (reset: E1)  ¬REL (reset: E1)			
E5	Send rollback to coordinator.			(reset: E1)		
E6	Send committed to coordinator.				CF (resync: E8)  ¬CF (committed: E7)	
E7	Receive forget from coordinator.				CF (resync: E8)  ¬CF&REL -Write FLR -CLOSE (reset: E1)	



Events		States				
		reset	prepare	rollback	committed	resync
					¬CF&¬REL -Write FLR (reset: E1)	
E8	Send resync to each in doubt participant to notify them of the outcome of the unit of work.					ACF (resync: E8) ¬ACF (reset: E1)
E9	Receive resync from each in doubt participant to determine the outcome of the unit of work.		-UNK		CF (resync) ¬CF&AFT -Write FLR (reset: E1)  CF (resync: E8) ¬CF -Write FLR (reset: E1)	

**Legend**

- ABORT      Rollback unit of work.
- CF          Conversation failure.
- CLOSE      Close connection to coordinator.
- CLR          Committed log record.
- CMT          Unit of work committed.
- COMMIT      Commit unit of work.
- FLR          Forget log record.
- NPT          No participants in current unit of work.
- REL          Release specified on prepare or request to commit.
- RBK          Unit of work rolledback.
- RRB          Receive rollback from coordinator.
- RRC          Receive request to commit from coordinator.
- UNK          Resync server has not received the outcome of the unit of work from the coordinator. Unit of work state is unknown.

## SEE ALSO

<b>insvar</b>	<i>PRCCNVCD</i> (on page 621)
<b>Semantic</b>	<i>LOGTSTMP</i> (on page 485)
	<i>RSYNCMGR</i> (on page 787)
	<i>SYNCCTL</i> (on page 915)
	<i>SYNCPTMGR</i> (on page 939)
	<i>SYNCRSY</i> (on page 982)

**NAME**

SYNCRRD — Sync Point Resync Reply

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'126D'

**Length** \*

**Class** CLASS

**Sprcls** COLLECTION - Collection Object

Sync Point Resync Reply (SYNCRRD) is used to inform the source SYNCPTMGR of the state of a unit of work at the target server. SYNCRRD is also used to acknowledge that a series of resync commands have been completed.

A resync reply object with *rsynctyp* equal to UOW (X'01') informs the source SYNCPTMGR of a unit of work state.

A resync reply object with *rsynctyp* equal to FORGET (X'02') is an acknowledgement that all resync commands have been completed.

**DSS Carrier: OBJDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'126D'	
<b>type</b>	INSTANCE_OF REQUIRED	RSYNCTYP - Resync Type
<b>uowid</b>	INSTANCE_OF OPTIONAL NOTE	UOWID - Unit of Work Identifier  This parameter is required when the value of parameter <i>rsynctyp</i> is X'01'. Otherwise, it must not be present.
<b>uowstate</b>	INSTANCE_OF OPTIONAL NOTE	UOWSTATE - Unit of Work State  This parameter is required when the value of parameter <i>rsynctyp</i> is X'01'. Otherwise, it must not be present.

**SEE ALSO**

**rpydta** SYNCRSY (on page 982)

**Semantic** SYNCPTOV (on page 944)

**NAME**

SYNCRSY — Sync Point Resync Command

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD  
**Codepoint** X'1069'  
**Length** \*  
**Class** CLASS  
**Sprcls** COMMAND - Command

Sync Point Resync Command (SYNCRSY) is used to resolve an in-doubt unit of work between two SYNCPTMGRs. The command may be sent from either the coordinator (or resync server) of the unit of work or the participant of an in-doubt unit of work.

**Source SYNCPTMGR**

The SYNCLOG data objects exchanged during the sync point operation are sent with this command to verify that the logs used to perform resync are consistent. For each unit of work requiring resynchronization, the source SYNCPTMGR sends a resync UOW command with the unit of work identifier and the state of the unit of work to the target SYNCPTMGR. If the SYNCPTMGR was the coordinator (or resync server), the state identifies the decision (commit or rollback). If the SYNCPTMGR was a participant, the state identifies an in-doubt state.

The source SYNCPTMGR sends a resync END command to inform the target server that all resync work has been completed and can be forgotten.

**DSS Carrier: RQSDSS**

**Target System Processing**

The target SYNCPTMGR replies to each resync UOW command with a resync UOW reply data object containing the unit of work identifier and its unit of work state.

The target SYNCPTMGR responds to the resync END command by sending the resync FORGET reply data object to indicate that all resync commands are processed and the commit decisions can be forgotten.

See the SYNCPTMGR state table in *SYNCP TOV* (on page 944) for more information on target system processing and reply messages.

**Exceptions**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'1069'	
<b>rsynctyp</b>	INSTANCE_OF REQUIRED	RSYNCTYP - Resync Type

	CMDTRG	
uowid	INSTANCE_OF OPTIONAL NOTE	UOWID - Unit of Work Identifier  This parameter is required if <i>rsynctyp</i> is set to UOW (X'01'). Otherwise, this parameter must not be present.
uowstate	INSTANCE_OF OPTIONAL NOTE	UOWSTATE - Unit of Work State  This parameter is required if <i>rsynctyp</i> is set to UOW (X'01'). Otherwise, this parameter must not be present.
clscmd	NIL	
inscmd	NIL	
<b>cmddta</b>		<b>COMMAND OBJECTS</b>
X'106F'	INSTANCE_OF OPTIONAL NOTE	SYNCLOG - Sync Point Log  When the parameter <i>rsynctyp</i> equals UOW (X'01'), exactly two occurrences of SYNCLOG must be sent. The source SYNCPTMGR SYNCLOG data object must be sent first, followed by the target SYNCPTMGR SYNCLOG data object. Otherwise, this object should not be sent.
<b>rpydta</b>		<b>REPLY OBJECTS</b>
X'126D'	INSTANCE_OF REQUIRED	SYNCRRD - Sync Point Resync Reply
<b>cmdrpy</b>		<b>COMMAND REPLIES</b>
X'1232'	INSTANCE_OF	AGNPRMRM - Permanent Agent Error
X'1254'	INSTANCE_OF	CMDCHKRM - Command Check
X'1245'	INSTANCE_OF	PRCCNVRM - Conversational Protocol Error
X'1251'	INSTANCE_OF	PRMNSPRM - Parameter Not Supported
X'124C'	INSTANCE_OF	SYNTAXRM - Data Stream Syntax Error
X'1252'	INSTANCE_OF	VALNSPRM - Parameter Value Not Supported

**SEE ALSO**

<b>insvar</b>	<i>PRCCNVCD</i> (on page 621)
<b>Semantic</b>	<i>RSYNCMGR</i> (on page 787) <i>SYNCPTOV</i> (on page 944)

NAME

SYNCTYPE — Sync Point Operation Type

DESCRIPTION (Semantic)

**Dictionary** QDDBASD  
**Codepoint** X'1187'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

Synpoint operation (SYNCTYPE) specifies the type of syncpoint operation to be performed by a SYNCPTMGR or the type of XA request to be performed by the XAMGR.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>INSTANCE VARIABLES</b>	
length	5	
class	X'119F'	
value	INSTANCE_OF	BYTDR - An 8-bit Data Representation Value
	ENUVAL	X'00' None (CMNSYNCP - Reuse connection)
	ENUVAL	X'01' Prepare to commit
	ENUVAL	X'02' Migrate resync responsibility to resync server
	ENUVAL	X'03' Unit of work committed
	ENUVAL	X'04' Unit of work rolled back
	ENUVAL	X'05' Request to commit unit of work
	ENUVAL	X'06' Forget unit of work
	ENUVAL	X'08' Request log information
	ENUVAL	X'09' New unit of work
	ENUVAL	X'0A' Migrated resync responsibility
	ENUVAL	X'0B' End association with unit of work
	ENUVAL	X'0C' Return list of prepared and heuristically completed XIDs
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

SEE ALSO

**insvar** *FORGET* (on page 423)  
*SYNCCRD* (on page 913)  
*SYNCTL* (on page 915)

**NAME**

SYNERRCD — Syntax Error Code

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'114A'

**Length** \*

**Class** CLASS

**Sprcls** STRING - String

Syntax Error Code (SYNERRCD) String specifies the condition that caused termination of data stream parsing.

clsvar	NIL	
insvar	CLASS INSTANCE VARIABLES	
length	5	
class	X'114A'	
value	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	LENGTH	2
	ENUVAL	X'01'
	NOTE	DSS header length less than 6.
	ENUVAL	X'02'
	NOTE	DSS header length does not match the number of bytes of data found.
	ENUVAL	X'03'
	NOTE	DSS header C-byte not D0.
	ENUVAL	X'04'
	NOTE	DSS header f-bytes either not recognized or not supported.
	ENUVAL	X'05'
	NOTE	DSS continuation specified but not found. For example, DSS continuation is specified on the last DSS, and the SNA LU 6.2 communication facility returned the SEND indicator.
	ENUVAL	X'06'
	NOTE	DSS chaining specified but no DSS found. For example, DSS chaining is specified on the last DSS, and the SNA LU 6.2 communication returned the SEND indicator.
	ENUVAL	X'07'
	NOTE	Object length less than four. For example, a command parameter's length is specified as two, or a command's length is specified as three.
	ENUVAL	X'08'
	NOTE	Object length does not match the number of bytes of data found. For example, an RQSDSS with a length of 150 contains a command whose

	length is 125, or a SRVDGN parameter specifies a length of 200, but there are only 50 bytes left in the DSS.
ENUVAL NOTE	X'09' Object length greater than maximum allowed. For example, a RECCNT parameter specifies a length of ten, but the parameter is defined to have a maximum length of eight.
ENUVAL NOTE	X'0A' Object length less than minimum required. For example, a SVRCOD parameter specifies a length of five, but the parameter is defined to have a fixed length of six.
ENUVAL NOTE	X'0B' Object length not allowed. For example, a FILEXPDT parameter is specified with a length of 11, but this would indicate that only half of the hours field is present instead of the complete hours field.
ENUVAL NOTE	X'0C' Incorrect large object extended length field (see the description of DSS). For example, an extended length field is present, but it is only three bytes long when it is defined to be a multiple of two bytes in length.
ENUVAL NOTE	X'0D' Object codepoint index not supported. For example, a codepoint of 8032 is encountered, but X'8' is a reserved code point index.
ENUVAL NOTE	X'0E' Required object not found. For example, a CLEAR command does not have a <i>filnam</i> parameter present, or a MODREC command is not followed by a RECORD command data object.
ENUVAL NOTE	X'0F' Too many command data objects sent. For example, a MODREC command is followed by 2 RECORD command data objects, or a DELREC command is followed by a RECORD object.
ENUVAL NOTE	X'10' Mutually-exclusive objects present. For example, a CRTDIRF command specifies both a DCLNAM and FILNAM parameters.
ENUVAL NOTE	X'11' Too few command data objects sent. For example, an INSRECEF command that specified RECCNT(5) is followed by only 4 RECORD command data objects.



ENUVAL NOTE	X'12' Duplicate object present. For example, a LSTFAT command has two FILNAM parameters specified.
ENUVAL NOTE	X'13' Invalid request correlator specified. Use PRCCNVRM with PRCCNVCD of 04 or 05 instead of this error code. This error code is being retained for compatibility with Level 1 of the architecture.
ENUVAL NOTE	X'14' Required value not found.
ENUVAL NOTE	X'15' Reserved value not allowed. For example, a INSRECEF command specified a RECCNT(0) parameter.
ENUVAL NOTE	X'16' DSS continuation less than or equal to two. For example, the length bytes for the DSS continuation have a value of one.
ENUVAL NOTE	X'17' Objects not in required order. For example, a RECAL object contains a RECORD object followed by a RECNBR object which is not in the defined order.
ENUVAL NOTE	X'18' DSS chaining bit not b'1', but DSSFMT bit 3 set to b'1'.
ENUVAL NOTE	X'19' Previous DSS indicated current DSS has the same request correlator, but the request correlators are not the same.
ENUVAL NOTE	X'1A' DSS chaining bit not b'1', but error continuation requested.
ENUVAL MINLVL NOTE	X'1B' 2 Mutually-exclusive parameter values specified. For example, an OPEN command specified PRPSHD(TRUE) and FILSHR(READER).
ENUVAL NOTE	X'1D' Codepoint not valid command. For example, the first codepoint in RQSDSS either is not in the dictionary or is not a codepoint for a command.
REQUIRED ENUVAL MINLVL	X'1E' 7

NOTE            The *atmind* instance variable is not set to its default value of X'00' on an EXCSQLSTT command within an atomic EXCSQLSTT chain.

<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

**SEE ALSO**

<b>insvar</b>	<i>DSS</i> (on page 308) <i>EXCSQLSTT</i> (on page 381) <i>HEXSTRDR</i> (on page 429) <i>MINLVL</i> (on page 518) <i>PRCCNVCD</i> (on page 621) <i>PRCCNVRM</i> (on page 625) <i>RQSDSS</i> (on page 774) <i>SVRCOD</i> (on page 911) <i>SRVDGN</i> (on page 873) <i>SYNTAXRM</i> (on page 989)
<b>Semantic</b>	<i>CLASS</i> (on page 158) <i>QDDBASD</i> (on page 651) <i>QRYROWSET</i> (on page 697) <i>STRING</i> (on page 888)

**NAME**

SYNTAXRM — Data Stream Syntax Error

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'124C'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

Data Stream Syntax Error (SYNTAXRM) Reply Message indicates that the data sent to the target agent does not structurally conform to the requirements of the DDM architecture. The target agent terminated parsing of the DSS when the condition SYNERRCD specified was detected.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'124C'	
<b>codpnt</b>	INSTANCE_OF MINLVL OPTIONAL NOTE	CODPNT - Codepoint 3  Specifies the codepoint of the object that caused the syntax error.
<b>rdbnam</b>	INSTANCE_OF MINLVL OPTIONAL	RDBNAM - Relational Database Name 3
<b>reccnt</b>	INSTANCE_OF MINVAL OPTIONAL NOTE  NOTE  MINLVL	RECCNT - Record Count 0  Required for requests to insert multiple records in a file.  This parameter is not returned by commands that operate on RDBs.  3
<b>svrdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code  8 - ERROR - Error Severity Code
<b>synerrcd</b>	INSTANCE_OF REQUIRED	SYNERRCD - Syntax Error Code
<b>clscmd</b>	NIL	

---

**inscmd**            **NIL**

**SEE ALSO**

**cmdrpy**            *ACCRDB* (on page 42)  
                      *ACCSEC* (on page 52)  
                      *BGNATMCHN* (on page 107)  
                      *BGNBND* (on page 110)  
                      *BNDSQLSTT* (on page 136)  
                      *CLSQRY* (on page 165)  
                      *CNTQRY* (on page 222)  
                      *DRPPKG* (on page 293)  
                      *DSCRDBTBL* (on page 300)  
                      *DSCSQLSTT* (on page 304)  
                      *ENDATMCHN* (on page 333)  
                      *ENDBND* (on page 336)  
                      *EXCSAT* (on page 363)  
                      *EXCSQLIMM* (on page 371)  
                      *EXCSQLSET* (on page 377)  
                      *EXCSQLSTT* (on page 381)  
                      *INTRDBRQS* (on page 445)  
                      *OPNQRY* (on page 555)  
                      *PRPSQLSTT* (on page 636)  
                      *RDBCMM* (on page 728)  
                      *RDBRLLBCK* (on page 745)  
                      *REBIND* (on page 753)  
                      *SECCHK* (on page 800)  
                      *SYNCCTL* (on page 915)  
                      *SYNCRSY* (on page 982)

**rpydta**            *COMMAND* (on page 240)

**Semantic**        *CMNTCPIP* (on page 214)  
                      *DSS* (on page 308)  
                      *QRYROWSET* (on page 697)  
                      *SYNCPTOV* (on page 944)  
                      *TCPSRCER* (on page 1013)  
                      *TCPTRGER* (on page 1015)

**NAME**

TASK — Task

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

In a multi-programming or multi-processing environment, a Task is one or more sequences of instructions treated by a control program as an element of work to be accomplished by a computer. A Task is the basic logical unit of work from the standpoint of a control program.

In the DDM architecture, a task is representative of the application program. A multitasking control program can support multiple application programs (each treated as a task) running concurrently. Usually a single user may have multiple application programs running under a single user identifier. Some of these programs may be running in the foreground (interactive level) while others may be running in the background (batch level). In addition, some control programs support multiple users active at one time, each with their own set of application programs. An application program can be broken into subtasks. Each subtask may or may not maintain contact with the starting task. Thus, on one system multiple tasks may represent a single user each performing some function for the user.

Within the DDM architecture, security, locking, and concurrency control are based on the task.

**SEE ALSO****Semantic** *CONCEPTS* (on page 243)

**NAME**

TCPCMNFL — TCP/IP Communications Failure

**DESCRIPTION (Semantic)**

**Dictionary** QDDTTRD

**Length** \*

**Class** HELP

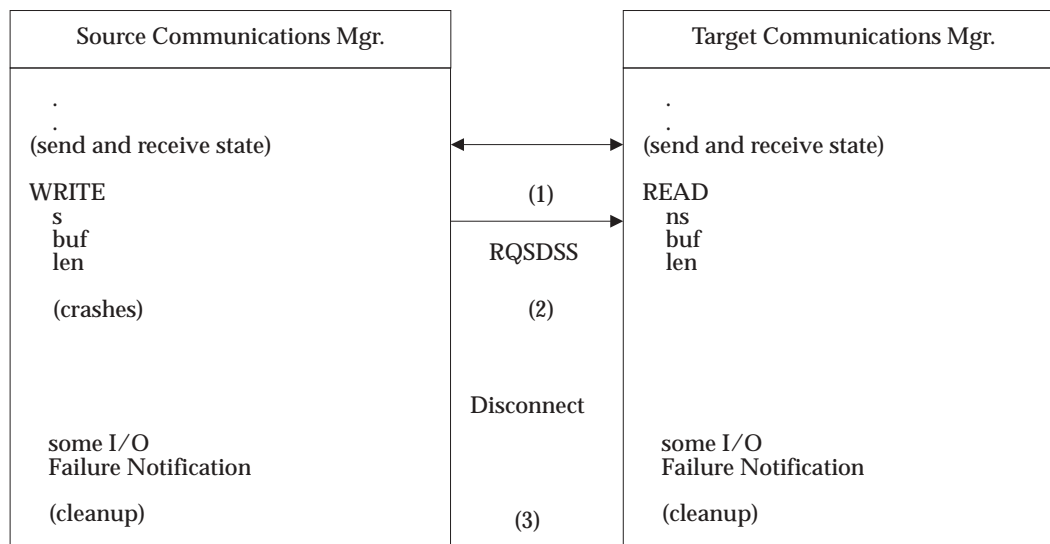
This term illustrates the communications sequence that occurs when a TCP/IP communication failure prematurely terminates the communications between the source communications manager (SCM) and the target communications manager (TCM). This can be caused by a TCP/IP connection protocol error, a connection outage, a communications line failure, a modem failure, a remote system failure, or failures of many other types. The result is that the SCM and TCM cannot communicate. Do not confuse communication failures with DDM-detected errors that result in a reply message. See the references in *TCPIPOVR* (on page 1000) for more information regarding TCP error handling.

The basic sequence for handling a communications failure is for both the SCM and the TCM to close their connections and perform any required cleanup. See Figure 3-110.

A sample protocol sequence is shown below.

The following assumption has been made for the sample protocol sequence:

- A connection has been successfully established between the SCM and the TCM (see *TCPCMNI* (on page 994)).



**Figure 3-110** Communications Failure Between Source and Target

**Figure Notes**

1. The SCM and TCM are passing RQSDSS and RPYDSS structures to each other. The SCM issues a WRITE call to send a RQSDSS to the TCM. The TCM receives the RQSDSS from the READ call. The *s* and *ns* parameters contain socket numbers at SCM and TCM respectively. The *buf* parameter contains the message and the *len* parameter contains the length of the message.
2. A communications failure occurs. TCP at SCM crashes and the communication link is disconnected.
3. Both ends know that TCP/IP connection is disconnected and they must perform any required cleanup functions. In general, these include:
  - File and database recovery at the server
    - For servers that support files, this includes:
      - Completing current DDM command processing, if possible
      - Releasing all record and stream locks that are being held
      - Closing any files that are open
      - Releasing all file locks that are being held
      - Performing any required additional cleanup, such as freeing up DCLFIL, DCLNAMs in the DCLFIL collection
    - For servers that support relational databases, this includes:
      - Performing a rollback on any accessed RDBs
      - Destroying any target SQLAM manager instances
      - Destroying any target SQLDM manager instances
      - Performing any additional cleanup required
  - Performing any required additional cleanup, such as cleaning up internal tables or control blocks, logging the failure, and so on

**SEE ALSO**

<b>Semantic</b>	<i>CMNTCPIP</i> (on page 214)
	<i>TCPCMNT</i> (on page 997)
	<i>TCPIPOVR</i> (on page 1000)

**NAME**

TCPCMNI — TCP/IP Communications Initiation

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

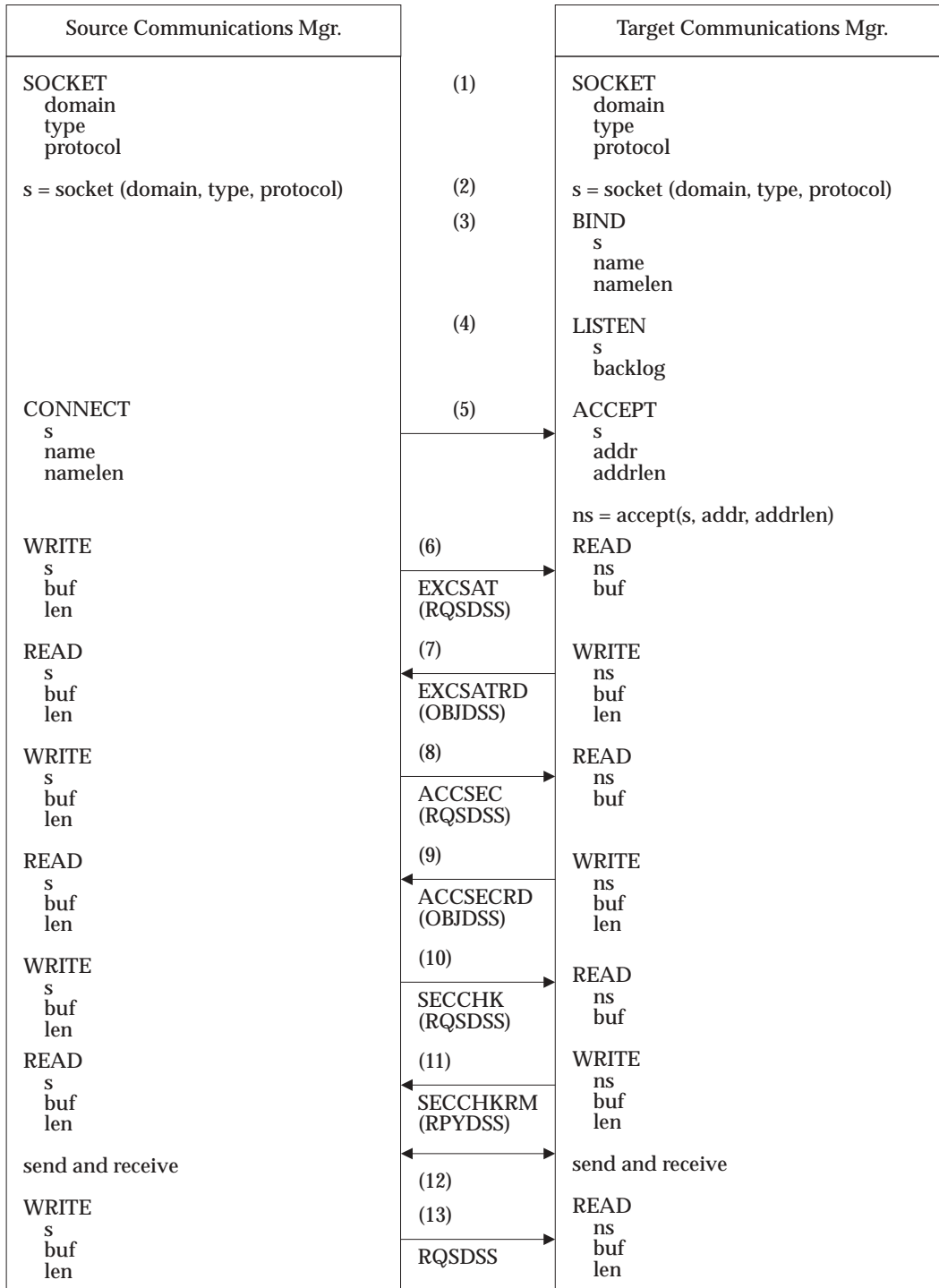
This term illustrates the use of TCP/IP communication facilities to initiate source-to-target communications. TCP/IP connection initiation is the responsibility of the TCP/IP communications facility. More information about TCP/IP connection initiation can be found in the referenced documents by Douglas Corner and *TCP/IP Tutorial and Technical Overview*, GG24-3376.

A sample protocol sequence is shown below. TCP/IP functions provide such an extensive set of functional capabilities that it is impossible to show all of the possible cases. Notes describing key points in the sequence follow the sequence diagram in Figure 3-111 (on page 995).

The following assumptions have been made for the sample protocol sequence:

- No connection exists between the SCM and the TCM.
- No error situations occur.
- Note that the full-duplex environment allows TCM and SCM to send and receive messages simultaneously. However, at this stage, DDM communications message exchanging is still in half-duplex mode until after the exchange of server attribute completed at (10).





**Figure 3-111 TCP/IP Communications Initiation by Source System**

**Figure Notes**

1. The SCM establishes communications with the TCM by issuing a SOCKET call. The *domain* parameter specifies the communications domain within which the communications will take place (for example, AF\_UNIX or AF\_NET). The *type* parameter specifies the semantics of communication which is SOCK\_STREAM representing byte stream socket. The *protocol* parameter specifies the selected protocol used is TCP.
2. Return values from successful completion of the SOCKET calls are the socket descriptor *s* which is used in subsequent TCP/IP calls. The value of -1 is returned if the SOCKET call fails.
3. At the TCM, the BIND call is issued using the socket descriptor *s* returned from the successful completion of the SOCKET call. The *name* parameter is the name to be assigned to the socket which include three fields in Internet protocol: *sin\_family*, *sin\_port*, and *sin\_addr* (not shown here). The *namelen* parameter is the length of this name in bytes.
4. At the TCM, after the BIND call is completed successfully, a LISTEN call is issued to ask TCP to queue connection for TCM. The *backlog* parameter defines the maximum length for the queue of pending connections.
5. The SCM issues a CONNECT call to establish the connection with the TCM. The TCM issues the ACCEPT call to accept the connection request from SCM. The *addr* parameter is a parameter that contains the address of the connecting entity. If the ACCEPT call is successful, the return value is the new socket value (*ns*) which is used in subsequent data transmissions with SCM.
6. The SCM issues another WRITE call to send the EXCSAT command in a RQSDSS to TCM to identify itself and its capabilities.
7. The TCM issues a WRITE call to reply EXCSATRD (an OBJDSS) to SCM.
8. The SCM issues a WRITE call to send the ACCSEC command in a RQSDSS to TCM to set up the security mechanisms.
9. The TCM issues a WRITE call to reply ACCSECRD (an OBJDSS) to SCM.
10. The SCM issues a WRITE call to send the SECCHK command in a RQSDSS to SCM to authenticate the user.
11. The TCM issue a WRITE call to reply SECCHKRM (an RPYDSS) to SCM.
12. At this point, communications have been established with the target system, the TCM process has been initiated, and DDM commands have begun to flow.
13. The SCM issue a WRITE call to send RQSDSS to SCM.

**SEE ALSO**

<b>Semantic</b>	<i>CMNTCPIP</i> (on page 214)
	<i>TCPCMNFL</i> (on page 992)
	<i>TCPCMNT</i> (on page 997)
	<i>TCPIPOVR</i> (on page 1000)
	<i>TCPSRCCD</i> (on page 1007)
	<i>TCPSRCCR</i> (on page 1010)
	<i>TCPSRCER</i> (on page 1013)
	<i>TCPTRGER</i> (on page 1015)

**NAME**

TCPCMNT — TCP/IP Communications Termination

**DESCRIPTION (Semantic)**

**Dictionary** QDDTTRD

**Length** \*

**Class** HELP

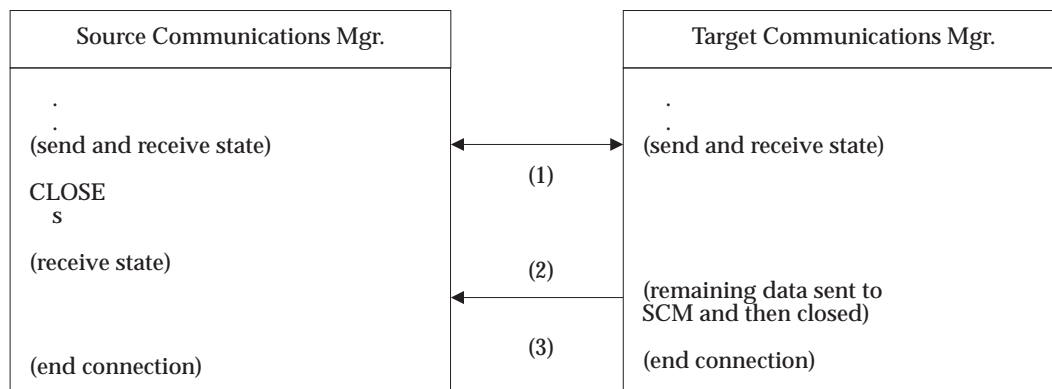
This term illustrates normal communication termination by communication manager.

Under normal circumstances, only the source communications manager (SCM) can terminate the connection between the SCM and the target communications manager (TCM). For termination in abnormal circumstances, see term TCPCMNFL. The SCM should terminate the connection only when the source system has completed all of its work with the target system. In order to terminate TCP/IP gracefully, SCM or TCM issues the TCP/IP CLOSE call. When an application at SCM tells TCP that it has no more data to send, it closes the connection by setting the FIN bit in the control field of the TCP segment header and transmits the remaining data. Upon receiving the FIN bit, TCP acknowledges that SCM has no more data to send and closes the SCM send connection. However, since TCP/IP is full duplex, only SCM is prohibited from sending any more data. The remote TCM may continue to send more data and the SCM still accepts data until there is no more data to send and TCP terminates the connection after it finishes sending data or when the TCM issues the CLOSE call. More detail of TCP/IP termination can be found in *UNIX Networking*, pp 64-89.

A sample protocol sequence using the CLOSE call is shown below. Notes describing key points in the sequence follow the sequence diagram.

The following assumptions have been made for the sample protocol sequence:

- A connection has been successfully established between the SCM and the TCM (see *TCPCMNI* (on page 994)).
- No error situation occurs.
- Note that the full duplex environment allows SCM and TCM to send and receive messages simultaneously.



**Figure 3-112** Normal Communications Termination

**Figure Notes**

1. The SCM issues a CLOSE call to inform TCP/IP that the connection is closed. Parameter *s* is the address of the socket that SCM connects to TCP/IP.
2. While SCM is sending a close call to TCP/IP, remaining data, if any, is sent to SCM simultaneously.
3. When no more data is sent by the remote TCM, the TCP at TCM closes the connection at that end.

**SEE ALSO**

**Semantic**            *CMNTCPIP* (on page 214)  
                          *TCPIPOVR* (on page 1000)

**NAME**

TCPHOST — TCP/IP Domain Qualified Host Name

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'11DD'

**Length** \*

**Class** CLASS

**Sprcls** SCALAR - Scalar Object

The domain qualified host name (TCPHOST) is used to determine a list of IP addresses supported by the CMNTCPIP host.

A source server resolves the target server's host name to an IP address (for example, gethostbyname) which is used to open a conversation to the target server. It must be in the form of a domain qualified name such as <rdbhost.drda.com>.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'11DD'	
value	INSTANCE_OF	CHRSTRDR - Character String
	MAXLEN	256
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** SYNCLOG (on page 922)  
 TCPPTHOST (on page 1006)

**NAME**

TCPIPOVR — TCP/IP Overview

**DESCRIPTION (Semantic)**

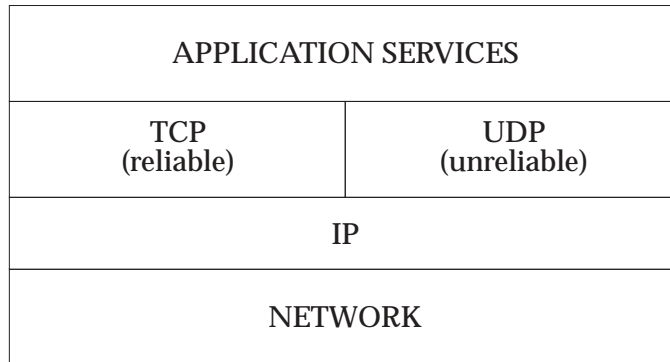
**Dictionary** QDDTTRD

**Length** \*

**Class** HELP

The Transmission Control Protocol/Internet Protocol (TCP/IP) is a reliable host-to-host protocol between hosts in packet-switched and in interconnected computer communications networks.

TCP/IP Internet is made up of several parts that interact to provide Services to the Internet users. The parts are Applications Services, TCP, UDP, IP, and Network. These parts and their relationship to each other are graphically displayed in Figure 3-113.



**Figure 3-113** TCP/IP Internet Components

**Reliable Stream Transport Service (TCP)**

TCP is a connection-oriented protocol designed to fit into a layered hierarchy of protocols which support multi-network applications. The TCP also provides reliable inter-process communication between pairs of processes in host computers attached to distinct but interconnected computer communication networks. In principle, the TCP should be able to operate above a wide spectrum of communication systems ranging from hard-wired connections to packet-switched or circuit-switched networks.

The reliable stream transport service is the level of service that DDM would need to provide the integrity required by DDM services. TCP services on top of IP provide the required functions.

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.

**User Datagram Protocol (UDP)**

UDP is an unreliable connectionless delivery service using IP to transport messages among machines. It adds the ability to distinguish among multiple destinations within a given host computer.

The DDM communication manager does not use or support UDP.

**Internet Protocol (IP)**

The Internet Protocol layer provides the unreliable, connectionless delivery system. It is unreliable because the delivery is not guaranteed. The packet may be lost, duplicated, or delivered out of order, but IP will not detect it or inform the sender or receiver. It is connectionless because each packet is treated independently from all others.

**Note:** These features of IP should not be of concern to DDM, because the TCP service provides the needed functions that are lacking in IP.

IP defines the basic unit of transfer and the exact format of all non-user data as it passes through the network. The basic transfer unit is called an Internet datagram. The datagram is divided into a header and data areas. The header contains the source and destination addresses, IP protocol version, and other control information. The datagram is then encapsulated in a network frame for navigation through the network.

**TCP/IP Applications**

The application services part is made up of high-level and specific services for applications. These services are built on UDP, TCP, or both. Examples of these services are:

- Telnet

Telnet provides services to allow a user or application at one site to access the login server at another. It then passes keystroke information from the local machine to the remote machine.

- File Transfer Protocol (FTP)

FTP allows users or applications to access a remote system, identify themselves, list remote directories, copy files to or from the remote machine, and execute a few simple commands remotely.

- Network File System (NFS)

The Sun NFS protocol enables machines to share file systems across a network. The NFS is designed to be machine and protocol independent. This is achieved through implementation on top of Sun's Remote Procedure Call (RPC). The machine independence in RPC is established through the use of External Data Representation (XDR) convention. NFS allows authorized users to access files located on remote systems as if they were local through RPC applications using UDP.

Currently DDM does not use or support NFS.

**Interfaces**

The TCP/Internet interface provides calls to send and receive datagrams addressed to TCP modules in hosts anywhere in the Internet system. These calls have parameters for passing the address, type of service, precedence, security, and other control information.

**Internet Addressing**

An internetwork address (IP address) is a 32-bit address field broken up into two parts: the network address and the local address. The number of bits for each field is defined by the number of local addresses *versus* the network addresses needed. The more local addresses defined will cut down on the number of network addresses available. Addressing is specified in classes A, B, C, and D. Table 3-21 shows the bit structure of the four classes of Internet addresses.

**Table 3-21** Internet Address Structure

Class Designation and Number of Bits	Number of Network Bits	Number of Local Bits
A -- 1	7	24
B -- 2	14	16
C -- 3	21	8
D -- 4	28	0

Figure 3-114 represents the structure of a class C address.

0 1 2	3 4 5 6 7 8 9 <sup>1</sup> 0 1 2 3 4 5 6 7 8 9 <sup>2</sup> 0 1 2 3	4 5 6 7 8 9 <sup>3</sup> 0 1
1 1 0	Network	Local

**Figure 3-114** Class C Internetwork Address

**Connection Establishment**

To identify the separate data streams that a TCP may handle, the TCP provides a port identifier. Since port identifiers are selected independently by each TCP they might not be unique. To provide for unique addresses within each TCP, an Internet address is concatenated with a port identifier to create a socket which will be unique throughout all networks connected together.

A connection is fully specified by the pair of sockets at the ends. A local socket may participate in many connections to different foreign sockets. A connection can be used to carry data in both directions; that is, it is *full duplex*.

TCP associates ports with processes through well-known sockets. Well-known sockets are a convenient mechanism for *a priori* associating a socket address with a standard service. For instance, the *Telnet-Server* process is permanently assigned to a particular socket, and other sockets are reserved for File Transfer, Remote Job Entry. A socket address might be reserved for access to a *Look-Up* service which would return the specific socket at which a newly created service would be provided. The concept of a well-known socket is part of the TCP specification, but the assignment of sockets to services is outside this specification.



**Closing a Connection**

Whenever an user decides to close a connection using the CLOSE call, the user who CLOSEs may continue to RECEIVE until he is told that the other side has CLOSED also. Thus, a program could initiate several SENDs followed by a CLOSE, and then continue to RECEIVE until signaled that the other side has CLOSED. TCP will signal a user, even if no RECEIVES are outstanding, that the other side has closed, so the user can terminate his side gracefully. A TCP will reliably deliver all buffers SENT before the connection was CLOSED so a user who expects no data in return need only wait to hear the connection was CLOSED successfully to know that all his data was received at the destination TCP. Users must keep reading connections they close for sending until the TCP says no more data.

**Basic Protocol Flow**

For descriptions of TCP/IP protocol flow refer to the following terms:

CMNTCPIP	TCP/IP Communication Manager (see <i>CMNTCPIP</i> (on page 214))
TCPCMNFL	TCP/IP Communications Failure (see <i>TCPCMNFL</i> (on page 992))
TCPCMNT	TCP/IP Communications Termination (see <i>TCPCMNT</i> (on page 997))
TCPSRCCD	TCP/IP Source Command with Data (see <i>TCPSRCCD</i> (on page 1007))
TCPSRCCR	TCP/IP Source Command Returning Data (see <i>TCPSRCCR</i> (on page 1010))
TCPSRCER	TCP/IP Source Detected Error (see <i>TCPSRCER</i> (on page 1013))
TCPTRGER	TCP/IP Target Detected Error (see <i>TCPTRGER</i> (on page 1015))

**Terminology****Connection**

A logical communication path identified by a pair of sockets.

**Datagram**

A message sent in a packet switched computer communications network.

**Destination Address**

The destination address, usually the network and host identifiers.

**Directory**

The directory component in the figure represents some directory service. The directory may be local or remote to the requester of the service. The directory is used to acquire IP addresses, security information such as what type of security the IP address supports, and the security level for the security type that is supported, and for any other information needed to help build the setup calls at the requester.

**Fragment**

A portion of a logical unit of data, in particular an Internet fragment is a portion of an Internet datagram.

**FTP**

A file transfer protocol.

**Header**

Control information at the beginning of a message, segment, fragment, packet, or block of data.

**Host**

A computer. In particular a source or destination of messages from the point of view of the communication network.

**Internet Address**

A source or destination address specific to the host level.

**Internet Datagram**

The unit of data exchanged between an Internet module and the higher-level protocol together with the Internet header.

**Internet Fragment**

A portion of the data of an Internet datagram with an Internet header.

**IP**

Internet Protocol.

**Octet**

An 8-bit byte.

**Packet**

A package of data with a header which may or may not be logically complete. More often a physical packaging than a logical packaging of data.

**Port**

The portion of a socket that specifies which logical input or output channel of a process is associated with the data.

**Process**

A program in execution. A source or destination of data from the point of view of the TCP or other host-to-host protocol.

**Segment**

A logical unit of data, in particular a TCP segment is the unit of data transferred between a pair of TCP modules.

**Socket**

An address which specifically includes a port identifier; that is, the concatenation of an Internet Address with a TCP port.

**Source Address**

The source address, usually the network and host identifiers.

**TCB**

Transmission control block, the data structure that records the state of a connection.

**TCP**

Transmission Control Protocol, a host-to-host protocol for reliable communication in internetwork environments.

**UDP**

User Datagram Protocol, a host-to-host protocol for unreliable, connectionless communication in internetwork environments.

**Well-known Port**

The well-known port is the socket on the server that accepts all the connection requests from the requester sockets. The well-known port is registered so its identity is known to provide a particular service. All connections to the server are initiated through this port. A connection request on the well-known port will cause TCP, through *ACCEPT* processing, to create a new socket, bind it to an available port, and associate this new socket and port to

the connecting requester. This frees up the well-known port to receive other connection requests.

**SEE ALSO**

**Semantic**      *CMNTCPIP* (on page 214)

**NAME**

TCPPORTHOST — TCP/IP Port Number and Domain-qualified Host Name

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD  
**Codepoint** X'1912'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

TCP/IP Port Number and Domain-Qualified Host Name (TCPPORTHOST) contains the port number and host name for a source or target server. The first two bytes contain the binary TCP/IP port number. The remaining bytes contain the TCP/IP domain-qualified host name.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'1912'	
tcpport	INSTANCE_OF LENGTH REQUIRED NOTE	BINDR - Binary Number Field 16  Binary 16 representation of TCP port number.
tcphost	INSTANCE_OF MINLEN REQUIRED NOTE	TCPHOST - TCP/IP Domain-qualified Host Name 1  TCP/IP host name of CMNTCPIP.
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** ACCRDBRM (on page 48)  
 SRVLSRV (on page 876)  
**Semantic** SRVLSRV (on page 876)

**NAME**

TCPSRCCD — TCP/IP Source Command with Data

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

After the source and target systems connected and exchanged server attributes, they may want to exchange command and data. At this time, depending on the contents of the message sent by source, command or data or command with data, target may send back to source either reply message or data. Because the difference in message contents sent from source results in different type of messages received from target, it is necessary to illustrate different scenarios when the source sends command with data and when the source sends command without data.

This term illustrates the communications sequence that occurs when the source communications manager (SCM) sends a command with data (a RQSDSS chain consisting of one RQSDSS and one OBJDSS) to the target system that resulted in a single reply message (a RPYDSS which contains an RM) returned by the target communications manager (TCM). For example, the DDM command could be an INSRECxx command followed by an OBJDSS that contains the command data (record) being inserted into the file, it could be a MODREC command followed by an OBJDSS that causes an existing record being modified, or it could be an EXCSQLIMM command followed by an OBJDSS that contains the SQL statement being executed.

The basic sequence for the SCM is to issue a WRITE call to send the RQSDSS (DDM command) and the OBJDSS (command data).

A sample protocol sequence is shown in Figure 3-115 (on page 1008). This sequence is only an example. Notes describing key points in the sequence follow the sequence diagram.

The following assumptions have been made for the sample protocol sequence:

- A connection has been successfully established between the SCM and the TCM (see *TCPCMNI* (on page 994)).
- The response to the RQSDSS is a single RPYDSS.
- No error situations occur.

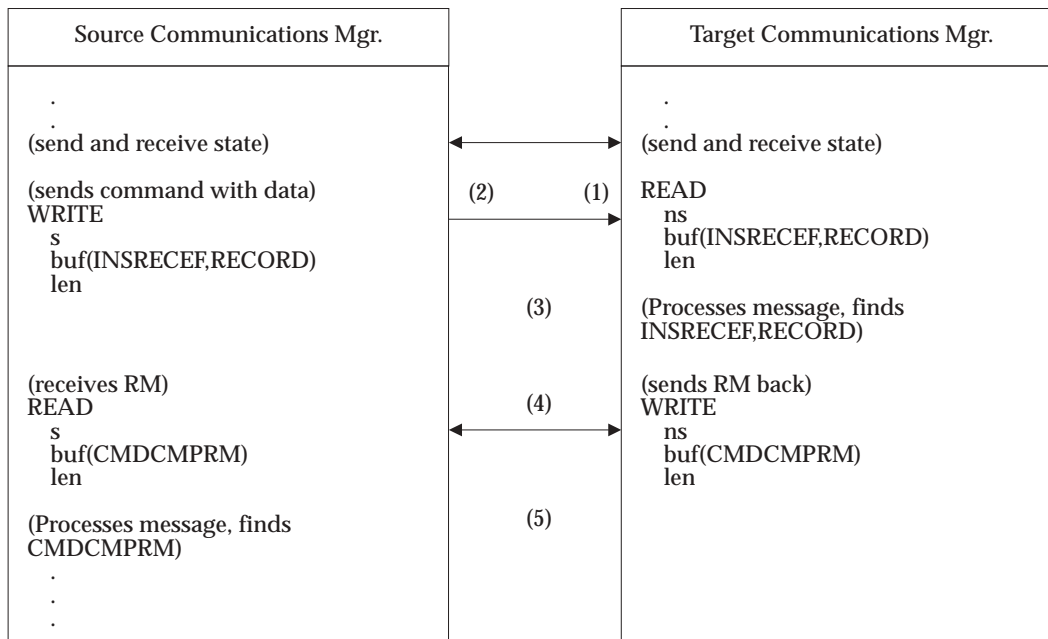


Figure 3-115 Source Sends Command With Data

**Figure Notes**

1. The TCM issues a READ call to receive data sent from the SCM. The *ns* parameter represents the socket that connects to SCM. The *buf* parameter contains data received (in this case RQSDSS and OBJDSS are anticipated). The *len* parameter contains the length of the message received.
2. The SCM issues a WRITE call to send a string of data containing a chained RQSDSS and OBJDSS. The RQSDSS contains an INSRECEF command, and the OBJDSS contains a record. The *s* parameter contains the socket number that connects to TCM. The *buf* parameter contains the chained RQSDSS and OBJDSS. The *len* parameter contains the length of the buffer sent.
3. The communication manager at TCM processes the messages, finds RQSDSS and OBJDSS and prepares a RPYDSS, in this case a CMDCMPRM reply message.
4. The TCM issues a WRITE call to send the RPYDSS to SCM. The RPYDSS is received by the READ call at SCM.
5. The SCM communication manager processes the messages and finds the CMDCMPRM.

This completes the example. The SCM and TCM are always in both receive and send state and the above sequence can be repeated for other DSS requests.

**SEE ALSO**

**Semantic**

*CMNTCPIP* (on page 214)  
*TCPIPOVR* (on page 1000)

**NAME**

TCPSRCCR — TCP/IP Source Command Returning Data

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

After the source and target systems connected and exchanged server attributes, they may want to exchange command and data. Depending on the contents of the message sent from source, that is command or data or command with data, the target may return to source data or reply message (RM). Because the difference in message contents sent from source results in a different type of messages received from target, it is necessary to illustrate different scenarios when the source sends command without data and when the source sends command with data.

This term illustrates the communications sequence that occurs when a source communications manager (SCM) sends a single command (a single RQSDSS) to the target communication manager (TCM) that results in data (one or more chained OBJDSSs) being returned to the SCM.

This is the normal protocol that is followed when the source agent retrieves records from a remote file or retrieves answer set data from a relational database. For example, the source agent sends a DDM command which requests that the target agent send a file record to the source agent. For example, the DDM command could be a GETREC or a SETxxx command.

A sample protocol sequence is shown below. Notes describing key points in the sequence follow the sequence diagram.

The following assumptions have been made for the sample protocol sequence:

- A connection has been successfully established between the SCM and TCM (see *TCPCMNI* (on page 994)).
- No error situations occur.



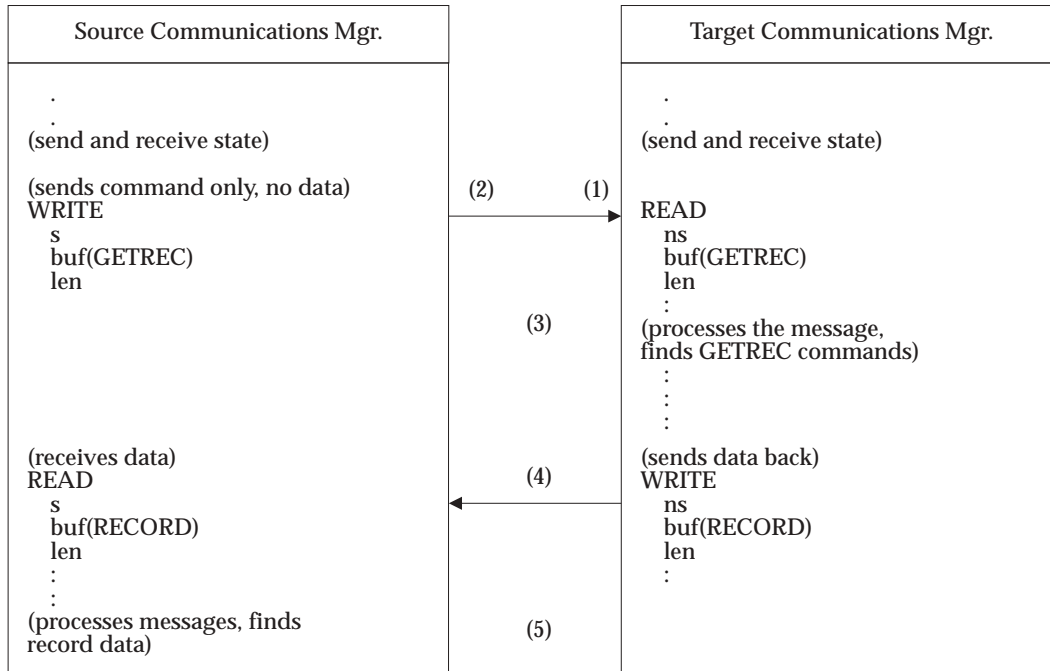


Figure 3-116 Source Sends Command With No Command Data

**Figure Notes**

1. The TCM issues a READ call to receive data sent from the SCM. The *ns* parameter represents the socket that connects to SCM. The *buf* parameter will store data received (in this case a RQSDSS which contains the GETREC command). The *len* parameter contains the length of the message received.
2. The SCM issues a WRITE call to send a RQSDSS DDM command. The *s* parameter contains the socket number that connects to TCM. The *buf* parameter contains the buffer which stores RQSDSS data. The *len* parameter contains the length of the buffer sent. For example, the RQSDSS may contain the GETREC command.
3. The communication manager at TCM processes the messages, finds RQSDSS which contains the GETREC command, and prepares an OBJDSS which contains record data.
4. The TCM issues a WRITE call to send the OBJDSS to SCM. The OBJDSS which contains a file record is received by the READ call at SCM.
5. The SCM communication manager processes the messages and finds the record in the OBJDSS.

This completes the example. The SCM and TCM are always in both receive and send state and the above sequence can be repeated many times for other DSS requests.

**SEE ALSO**

**Semantic**

*CMNTCPIP* (on page 214)

*TCPIPOVR* (on page 1000)

**NAME**

TCPSRCER — TCP/IP Source Detected Error

**DESCRIPTION (Semantic)**

**Dictionary** QDDTTRD

**Length** \*

**Class** HELP

When errors created by the DDM server are detected by either source or target communication manager, the TCP/IP connection is closed. In all instances, the source is the one that issues the close call. On the one hand, when the source detects errors created by the target DDM manager, it has to close the connection until the problems are fixed. On the other hand, when target detects errors created by the source DDM manager, it returns a syntax error reply message (SYNTAXRM) to the source so that the source may decide to close the connection.

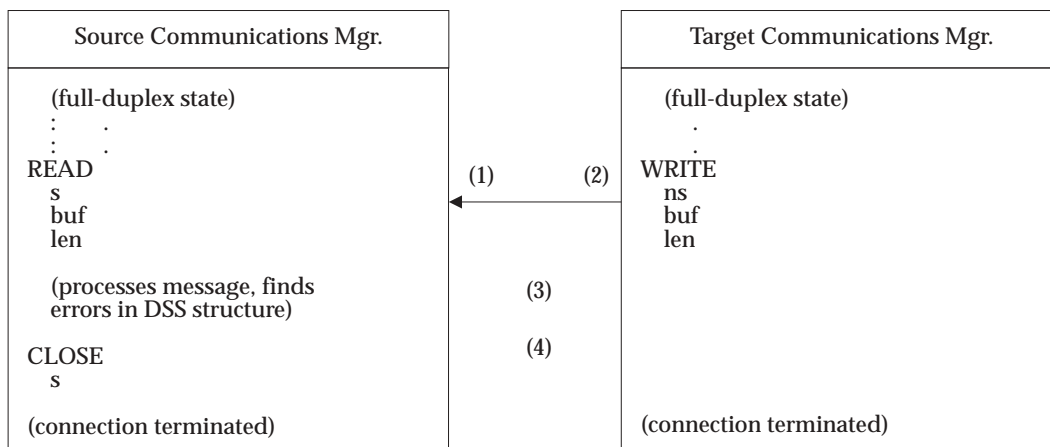
This term illustrates the communication sequence that occurs when the source DDM manager detects errors created by the target system and closes the connection. For a sequence that occurs when the target detect errors created by the source manager, refer to term TCPTRGER.

When the SCM processes the DSSs received from the TCM, an error may be detected by the SCM, the source agent, or some other source manager that prevents the DSS contents from being processed. The error might be that the TCM sent structures that are not expected by the SCM and thus SCM closes the TCP/IP connection.

A sample protocol sequence is shown below.

The following assumptions have been made for the sample protocol sequence:

- A connection has been successfully established between the SCM and TCM (see *TCPCMNI* (on page 994)).
- The initial conditions for this sequence are that the TCM processes the last RQSDSS received from the SCM and issues a WRITE call to transmit the RPYDSS to the SCM. However, it builds the RPYDSS structure incorrectly.



**Figure 3-117** Source System Detects Error

**Figure Notes**

1. The SCM READ call is waiting for a response from TCM. The *s* parameter contains the TCP/IP socket number that connects to TCM. The *buf* parameter contains data received from TCM and the *len* parameter contains the length of the message received.
2. The TCM issues an RPYDSS in response to the RQSDSS previously sent by the SCM. However, the RPYDSS structure contains errors. The *ns* parameter contains the new socket that connects to SCM and the *buf* parameter contains data sent to SCM.
3. The SCM processes the RPYDSS. However, in doing this the SCM encounters an error that prevents it from successfully processing the RPYDSS.
4. The action to be taken by the SCM after the detecting error in this case is to abort the connection. SCM issues a CLOSE call to close the connection and discard all messages remaining in TCP/IP. The *s* parameter contains the socket number.

**SEE ALSO**

**Semantic**            *CMNTCPIP* (on page 214)  
                          *TCPIPOVR* (on page 1000)

**NAME**

TCPTRGER — TCP/IP Target Detected Error

**DESCRIPTION (Semantic)****Dictionary** QDDTTRD**Length** \***Class** HELP

When errors created by the DDM server are detected by either the source or target communication manager, the TCP/IP connection must be terminated. In most instances, it is the source who issues the close of connection call. If possible, for unrecoverable errors, send an AGNPRMRM with a request correlator of FF, close the connection and terminate. On the one hand, if the DDM manager from the source detects errors created by the target DDM manager, it closes the connection and notifies the target of the errors. On the other hand, if the DDM manager from the target detects errors created by the source DDM manager, it must send a syntax error reply message (SYNTAXRM) to the source and let the source issue the close connection.

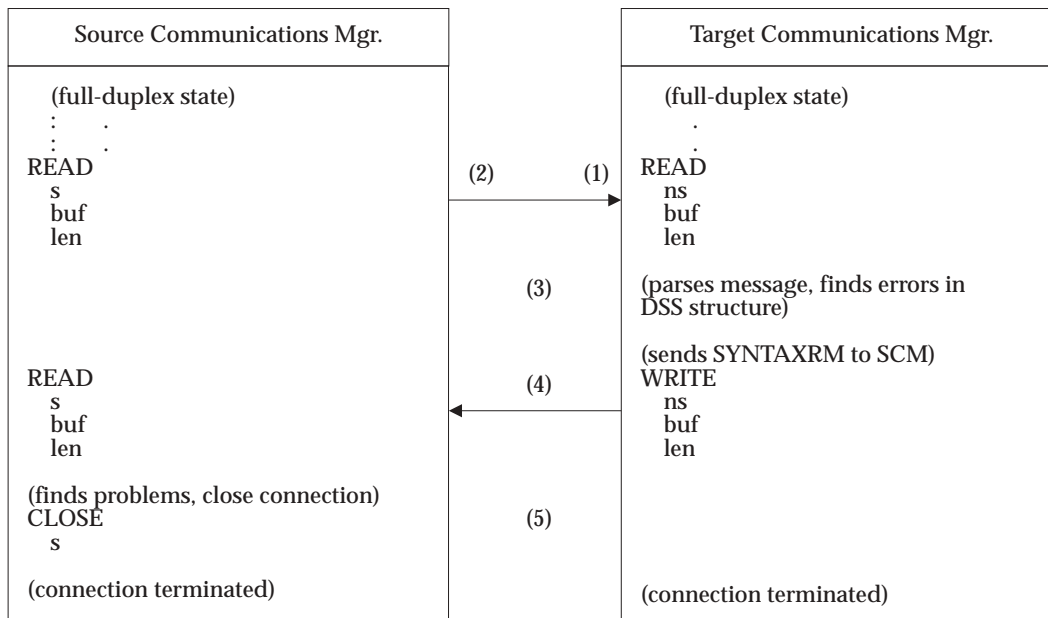
This term illustrates the communications sequence that occurs when a target DDM manager detects an error and reports it to the source communications manager (SCM). These errors are detected above the TCP/IP communication facility by the DDM server. This discussion assumes that the detected error results in a reply message with SVRCOD(ERROR) or higher, and if an RQSDSS chain is being processed, error continuation is *not* being performed.

When the TCM processes the data structures received from the SCM, an error may be detected that prevents the TCM or target agent from processing the data structure. The error might be that the SCM sent structures that were not expected by the TCM. In this case, the SCM must close the TCP/IP connection.

A sample protocol sequence is shown below. Notes describing key points in the sequence follow the sequence diagram.

The following assumptions have been made for the sample protocol sequence:

- A connection has been successfully established between the SCM and TCM (see *TCPCMNI* (on page 994)).



**Figure 3-118** Target System Detects Error

#### Figure Notes

1. The TCM `READ` call is waiting for a response from the TCM. The *ns* parameter contains the TCP/IP socket number that connects to the SCM. The *buf* parameter contains data received from the SCM and the *len* parameter contains the length of the message received.
2. The SCM issues a `WRITE` call to send an RQSDSS to the TCM. However, this RQSDSS structure contains errors. The *s* parameter contains the new socket that connects to the TCM and the *buf* parameter contains data sent to the TCM.
3. The TCM processes the RQSDSS. However, in doing this the TCM encounters an error that prevents it from successfully processing the RQSDSS.
4. The TCM sends a `SYNTAXRM` which specifies the type of error to the SCM.
5. The SCM learns that its message is not understood by the TCM and issues the `CLOSE` call to terminate the TCP/IP connection.

The source agent is responsible for performing any recovery actions that are necessary. The TCM may wish to log the error or notify someone on the local system, but this is not a requirement.

**SEE ALSO**

**Semantic**

*CMNTCPIP* (on page 214)  
*TCPIPOVR* (on page 1000)  
*TCPSRCER* (on page 1013)

**NAME**

TEXT — Text Character String

**DESCRIPTION (Semantic)****Dictionary** QDDBASD**Codepoint** X'114B'**Length** \***Class** CLASS**Sprcls** STRING - String

Text Character String (TEXT) specifies character string data that can be read.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'114B'	
value	INSTANCE_OF	CHRSTRDR - Character String
	MINLEN	0
	MAXLEN	255
	OPTIONAL	
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO****Semantic** *TITLE* (on page 1020)**semantic** *CODPNT* (on page 233)  
*CONSTANT* (on page 244)  
*HELP* (on page 425)



**NAME**

TIMEOUT - Time Out

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Length** \*

**Class** CLASS

**Sprcls** FIELD - A Discrete Unit of Data

The TIMEOUT parameter is a 64-bit binary number which measures time in milliseconds. It represents the elapsed time before the transaction is rolled back. The timer is started at the beginning of the transaction SYNCCTL(New UOW).

If the request SYNCCTL(Prepare to Commit) is not received during this time period, the transaction will be implicitly rolled back.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>INSTANCE VARIABLES</b>	
length	12	
class	X'1907'	
value	INSTANCE_OF LENGTH	BINDR - Binary Number Field 64
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** SYNCCTL (on page 915)

**NAME**

TITLE — Title

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'0045'

**Length** \*

**Class** CLASS

**Sprcls** TEXT - Text Character String

TITLE (TITLE) Text Character String specifies a brief description used in presentations of information about an architected term when used as a variable of a DDM architecture term.

When used as an attribute of an instance of an object, TITLE specifies a brief description of the specific object instance that the server can use in presentations about the objects that it manages.

When used as an attribute name, TITLE specifies that the attribute value is a brief description to be used when formatting the variable for printing or display.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	*	
class	X'0045'	
value	INSTANCE_OF	CHRSTRDR - Character String Data Representation
	MINLEN	0
	MAXLEN	255
	OPTIONAL	
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

- clscmd** CLASS (on page 158)
- clsvar** CLASS (on page 158)
- cmddda** COMMAND (on page 240)
- cmdrpy** COMMAND (on page 240)
- inscmd** CLASS (on page 158)
- insvar** BGNBND (on page 110)  
CLASS (on page 158)  
CODPNT (on page 233)  
CONSTANT (on page 244)  
HELP (on page 425)
- mgrdepls** MANAGER (on page 491)
- mgrlvln** MANAGER (on page 491)
- rpydta** COMMAND (on page 240)
- Semantic** CLASS (on page 158)  
REBIND (on page 753)

	<i>SCALAR</i> (on page 796)
<b>semantic</b>	<i>CLASS</i> (on page 158) <i>CODPNT</i> (on page 233) <i>CONSTANT</i> (on page 244) <i>HELP</i> (on page 425)
<b>sprcls</b>	<i>CLASS</i> (on page 158)
<b>vldattls</b>	<i>AGENT</i> (on page 61) <i>CCSIDMGR</i> (on page 150) <i>CMNAPPC</i> (on page 184) <i>CMNMGR</i> (on page 196) <i>CMNSYNCPT</i> (on page 202) <i>CMNTCPIP</i> (on page 214) <i>DICTIONARY</i> (on page 286) <i>FIELD</i> (on page 415) <i>MANAGER</i> (on page 491) <i>RDB</i> (on page 718) <i>RSYNCMGR</i> (on page 787) <i>SECMGR</i> (on page 814) <i>SERVER</i> (on page 824) <i>SQLAM</i> (on page 847) <i>SUPERVISOR</i> (on page 908) <i>SYNCPTMGR</i> (on page 939) <i>XAMGR</i> (on page 1063)

NAME

TRGNSPRM — Target Not Supported

DESCRIPTION (Semantic)

**Dictionary** QDDBASD  
**Codepoint** X'125F'  
**Length** \*  
**Class** CLASS  
**Sprcls** RPYMSG - Reply Message

Target Not Supported (TRGNSPRM) Reply Message indicates that the object specified as a command target parameter is not an object of a class that the target server supports. This condition can arise when a target server can address objects of classes that DDM or product extensions to DDM cannot support. It can also arise for valid DDM classes that the target server does not support.

For example, the TRGNSPRM is returned if the name of the object a FILNAM (command target) parameter specifies is either not a file (for instance, a program library) or is not of a DDM file class (for instance, a file class the target system does not support).

DSS Carrier: RPYDSS

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'125F'	
<b>objnam</b>	ENUCLS	RDBNAM - Relational Database Name
	NOTE	The RDBNAM is returned when the target object that is not supported is an RDB.
	MINLVL	3
	ENUCLS	FILNAM - File Name
	NOTE	The FILNAM is returned when the target object that is not supported is a file.
	MINLVL	4
	ENUCLS	DRCNAM - Directory Name
	NOTE	The DRCNAM is returned when the target object that is not supported is a directory.
	MINLVL	4
	ENUCLS	DCLNAM - Declared Name
	NOTE	The DCLNAM is returned when the target object that is not supported is referenced by a declare name.
	MINLVL	4
	ENUCLS	QUENAM - Queue Name
	NOTE	The QUENAM is returned when the target object that is not supported is a queue.

	MINLVL	4
	OPTIONAL	
srvdgn	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
svrcod	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code  8 - ERROR - Error Severity Code
clscmd	NIL	
inscmd	NIL	

**SEE ALSO**

<b>cmdrpy</b>	<i>ACCRDB</i> (on page 42) <i>ACCSEC</i> (on page 52) <i>BGNBND</i> (on page 110) <i>BNDSQLSTT</i> (on page 136) <i>CLSQR</i> Y (on page 165) <i>CNTQR</i> Y (on page 222) <i>DRPPKG</i> (on page 293) <i>DSCRDBTBL</i> (on page 300) <i>DSCSQLSTT</i> (on page 304) <i>ENDBND</i> (on page 336) <i>EXCSAT</i> (on page 363) <i>EXCSQLIMM</i> (on page 371) <i>EXCSQLSTT</i> (on page 381) <i>INTRDBRQS</i> (on page 445) <i>OPNQRY</i> (on page 555) <i>PRPSQLSTT</i> (on page 636) <i>RDBCMM</i> (on page 728) <i>RDBRLLBCK</i> (on page 745) <i>REBIND</i> (on page 753) <i>SECCHK</i> (on page 800)
<b>rpydta</b>	<i>COMMAND</i> (on page 240)
<b>Semantic</b>	<i>SUBSETS</i> (on page 902)

## NAME

TRUE — True State

## DESCRIPTION (Semantic)

**Dictionary** QDDPRMD**Codepoint** X'003B'**Length** \***Class** CONSTANT

True State (TRUE) Boolean State specifies the logical state of being TRUE. This constant is also used wherever only one of two states can be specified.

**Literal Form**

The literal for true is the capitalized TRUE.

---

value	X'F1'
-------	-------

## SEE ALSO

<b>insvar</b>	<i>BOOLEAN</i> (on page 142) <i>FORGET</i> (on page 423) <i>OUTEXP</i> (on page 571) <i>QRYROWNBR</i> (on page 694) <i>RDBALWUPD</i> (on page 726) <i>RDBCMTOK</i> (on page 731) <i>RLSCONV</i> (on page 765) <i>RTNSQLDA</i> (on page 795) <i>SQLCSRHLD</i> (on page 858) <i>SYNCCTL</i> (on page 915) <i>SYNERRCD</i> (on page 985)
<b>rpydta</b>	<i>PRPSQLSTT</i> (on page 636) <i>SYNCCTL</i> (on page 915)
<b>Semantic</b>	<i>BOOLEAN</i> (on page 142) <i>EXCSQLIMM</i> (on page 371) <i>EXCSQLSTT</i> (on page 381) <i>FORGET</i> (on page 423) <i>OUTEXP</i> (on page 571) <i>QRYROWNBR</i> (on page 694) <i>RDBALWUPD</i> (on page 726) <i>RTNSQLDA</i> (on page 795) <i>SQLCSRHLD</i> (on page 858) <i>SYNCCTL</i> (on page 915) <i>SYNCPTOV</i> (on page 944)

**NAME**

TYPDEF — Data Type to Data Representation Definition

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'0029'

**Length** \*

**Class** CLASS

**Sprcls** ORDCOL - Ordered Collection

Data Type to Data Representation Definition (TYPDEF) Ordered Collection defines and/or identifies how a manager represents the data type in a type family. The TYPDEFNAM either identifies a set of predefined definitions or specifies the name assigned to the mapping definitions specified in the FDODSC. The TYPFMLNM identifies the name of a *type family*, such as the SQL type family. When present, an FDODSC explicitly defines a set of data type to data representation mapping definitions.

**DSS Carrier: OBJDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'0029'	
<b>fdodsc</b>	INSTANCE_OF OPTIONAL	FDODSC - FD:OCA Data Descriptor
<b>typfmlnm</b>	INSTANCE_OF OPTIONAL	TYPFMLNM - Data Type Family Name
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**cmddda** *BNDSQLSTT* (on page 136)  
*DSCRDBTBL* (on page 300)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSET* (on page 377)  
*EXCSQLSTT* (on page 381)  
*OPNQRY* (on page 555)  
*PRPSQLSTT* (on page 636)

**insvar** *RDB* (on page 718)  
*SQLAM* (on page 847)

**rpydta** *BNDSQLSTT* (on page 136)  
*CLSQRY* (on page 165)  
*CNTQRY* (on page 222)  
*DRPPKG* (on page 293)  
*DSCRDBTBL* (on page 300)  
*DSCSQLSTT* (on page 304)  
*ENDBND* (on page 336)  
*EXCSQLIMM* (on page 371)

*EXCSQLSET* (on page 377)  
*EXCSQLSTT* (on page 381)  
*OPNQRY* (on page 555)  
*PRPSQLSTT* (on page 636)  
*RDBCMM* (on page 728)  
*RDBRLLBCK* (on page 745)  
*REBIND* (on page 753)

**Semantic**

*SQLAM* (on page 847)  
*TYPDEFNAM* (on page 1027)  
*TYPDEFOVR* (on page 1030)  
*TYPFMLNM* (on page 1033)



**NAME**

TYPDEFNAM — Data Type Definition Name

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD  
**Codepoint** X'002F'  
**Length** \*  
**Class** CLASS  
**Sprcls** NAME - Name

Data Type Definition Name (TYPDEFNAM) specifies the name of a data type definition. See *TYPDEF* (on page 1025).

When the TYPDEFNAM object is specified as a command/reply data object, the value specified applies to the following command data objects and reply data objects for that command, respectively. When TYPDEFNAM is repeatable, the value of one TYPDEFNAM object is applicable only to those objects (command data or reply data) that are sent before another TYPDEFNAM object is sent. The value of TYPDEFNAM that a command specifies is in effect only for that command. This rule applies to all commands, unless specified otherwise.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
<b>length</b>	*	
<b>class</b>	X'002F'	
<b>value</b>	INSTANCE_OF	NAMDR - Name Data
	ENUVAL	'QTDSQL370'
	NOTE	System/390 SQL type definition
	ENUVAL	'QTDSQL400'
	NOTE	Application System/400 SQL type definition
	ENUVAL	'QTDSQLX86'
	NOTE	Intel 80X86 SQL type definition
	ENUVAL	'QTDSQLASC'
	NOTE	General ASCII SQL type definition for machines characterized by the use of ASCII strings, IEEE floating-point numbers, and non-byte-reversed floating-point and integer numbers (for example, RS/6000).
	MINLVL	4
	ENUVAL	'QTDSQLVAX'
	NOTE	DEC VAX SQL type definition.
	MINLVL	4
	ENUVAL	'QTDSQLJVM'
	NOTE	JAVA SQL Type Definition
	MINLVL	7
	REQUIRED	
<b>clscmd</b>	<b>NIL</b>	

---

**inscmd**            **NIL**

**SEE ALSO**

**insvar**            *ACCRDB* (on page 42)  
                       *ACCRDBRM* (on page 48)

**cmdtda**            *BNDSQLSTT* (on page 136)  
                       *DSCRDBTBL* (on page 300)  
                       *EXCSQLIMM* (on page 371)  
                       *EXCSQLSET* (on page 377)  
                       *EXCSQLSTT* (on page 381)  
                       *OPNQRY* (on page 555)  
                       *PRPSQLSTT* (on page 636)

**rpydta**            *ACCRDB* (on page 42)  
                       *BGNATMCHN* (on page 107)  
                       *BGNBND* (on page 110)  
                       *BNDSQLSTT* (on page 136)  
                       *CLSQRY* (on page 165)  
                       *CNTQRY* (on page 222)  
                       *DRPPKG* (on page 293)  
                       *DSCRDBTBL* (on page 300)  
                       *DSCSQLSTT* (on page 304)  
                       *ENDATMCHN* (on page 333)  
                       *ENDBND* (on page 336)  
                       *EXCSQLIMM* (on page 371)  
                       *EXCSQLSET* (on page 377)  
                       *EXCSQLSTT* (on page 381)  
                       *OPNQRY* (on page 555)  
                       *PRPSQLSTT* (on page 636)  
                       *RDBCMM* (on page 728)  
                       *RDBRLLBCK* (on page 745)  
                       *REBIND* (on page 753)

**Semantic**        *ACCRDB* (on page 42)  
                       *BGNBND* (on page 110)  
                       *BNDSQLSTT* (on page 136)  
                       *CLSQRY* (on page 165)  
                       *CNTQRY* (on page 222)  
                       *DRPPKG* (on page 293)  
                       *DSCRDBTBL* (on page 300)  
                       *DSCSQLSTT* (on page 304)  
                       *ENDBND* (on page 336)  
                       *EXCSQLIMM* (on page 371)  
                       *EXCSQLSET* (on page 377)  
                       *EXCSQLSTT* (on page 381)  
                       *FIXROWPRC* (on page 417)  
                       *LMTBLKPRC* (on page 475)  
                       *OPNQRY* (on page 555)  
                       *PRPSQLSTT* (on page 636)  
                       *RDBCMM* (on page 728)  
                       *RDBRLLBCK* (on page 745)  
                       *REBIND* (on page 753)

*TYPDEF* (on page 1025)  
*TYPDEFOVR* (on page 1030)

## NAME

TYPDEFOVR — TYPDEF Overrides

## DESCRIPTION (Semantic)

**Dictionary** QDDPRMD**Codepoint** X'0035'**Length** \***Class** CLASS**Sprcls** COLLECTION - Collection Object

TYPDEF Overrides (TYPDEFOVR) specifies the Coded Character Set Identifiers (CCSIDs) that are in a named TYPDEF. The definer of the TYPDEF is responsible for defining how these CCSIDs are used in conjunction with the TYPDEFNAM. The definer of the TYPDEF is also responsible for defining which CCSIDs are valid for each of the CCSIDxBC parameters. If the current command's command data object does not specify one or more of the CCSIDxBC parameters, then the values of the unspecified CCSIDxBCs are inherited from the previously received TYPDEFOVR CCSIDxBCs for this command. If no previous TYPDEFOVR CCSIDxBC has ever been received, the corresponding TYPDEFOVR CCSID is not used. The values of TYPDEFOVR CCSIDxBCs a command specifies are in effect only for that command.

When the TYPDEFOVR object is specified as a command/reply data object, at least one of the CCSIDxBCs must be specified, and the values specified for the CCSIDxBC parameters apply to the following command data objects and reply data objects for that command, respectively. When TYPDEFOVR is repeatable the value of one TYPDEFOVR object is applicable to only those objects (command data or reply data) that are sent before another TYPDEFOVR object is sent. This rule applies to all commands, unless specified otherwise.

Figure 3-119 (on page 1031) illustrates TYPDEFOVR.

Timestamps	Commands/ Replies	CCSIDs	
		specified	in effect
	ACCRDB/RM	SA DA MA	SA DA MA
t1	begin	St1	St1 DA MA
t2	C / O R M E M P A L N Y D	Dt2	St1 Dt2 MA
t3		St3 Mt3	St3 Dt2 Mt3
t4	end	Dt4	St3 Dt4 Mt3
t5			SA DA MA (Reset)

t1, t2, t3, t4, t5: timestamps in increasing order  
 SA, St1, St2, St3, St4, St5: CCSIDSBCs (single-byte chars.)  
 DA, Dt1, Dt2, Dt3, Dt4, Dt5: CCSIDDBCs (double-byte chars.)  
 MA, Mt1, Mt2, Mt3, Mt4, Mt5: CCSIDMBCs (mixed-byte chars.)

Figure 3-119 The Usage of TYPDEFOVR

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'0035'	
<b>ccsiddbc</b>	INSTANCE_OF OPTIONAL	CCSIDDBC - CCSID for Double-byte Characters
<b>ccsidmbc</b>	INSTANCE_OF OPTIONAL	CCSIDMBC - CCSID for Mixed-byte Characters
<b>ccsidsbc</b>	INSTANCE_OF OPTIONAL	CCSIDSBC - CCSID for Single-byte Characters
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

## SEE ALSO

<b>cmdtta</b>	<i>BNDSQLSTT</i> (on page 136) <i>DSCRDBTBL</i> (on page 300) <i>EXCSQLIMM</i> (on page 371) <i>EXCSQLSET</i> (on page 377) <i>EXCSQLSTT</i> (on page 381) <i>OPNQRY</i> (on page 555) <i>PRPSQLSTT</i> (on page 636)
<b>insvar</b>	<i>ACCRDB</i> (on page 42) <i>ACCRDBRM</i> (on page 48)
<b>rpydta</b>	<i>ACCRDB</i> (on page 42) <i>BGNATMCHN</i> (on page 107) <i>BGNBND</i> (on page 110) <i>BNDSQLSTT</i> (on page 136) <i>CLSQRY</i> (on page 165) <i>CNTQRY</i> (on page 222) <i>DRPPKG</i> (on page 293) <i>DSCRDBTBL</i> (on page 300) <i>DSCSQLSTT</i> (on page 304) <i>ENDATMCHN</i> (on page 333) <i>ENDBND</i> (on page 336) <i>EXCSQLIMM</i> (on page 371) <i>EXCSQLSET</i> (on page 377) <i>EXCSQLSTT</i> (on page 381) <i>OPNQRY</i> (on page 555) <i>PRPSQLSTT</i> (on page 636) <i>RDBCMM</i> (on page 728) <i>RDBRLLBCK</i> (on page 745) <i>REBIND</i> (on page 753)
<b>Semantic</b>	<i>BGNBND</i> (on page 110) <i>BNDSQLSTT</i> (on page 136) <i>CLSQRY</i> (on page 165) <i>CNTQRY</i> (on page 222) <i>DRPPKG</i> (on page 293) <i>DSCRDBTBL</i> (on page 300) <i>DSCSQLSTT</i> (on page 304) <i>ENDBND</i> (on page 336) <i>EXCSQLIMM</i> (on page 371) <i>EXCSQLSET</i> (on page 377) <i>EXCSQLSTT</i> (on page 381) <i>FIXROWPRC</i> (on page 417) <i>LMTBLKPRC</i> (on page 475) <i>OPNQRY</i> (on page 555) <i>PRPSQLSTT</i> (on page 636) <i>RDBCMM</i> (on page 728) <i>RDBRLLBCK</i> (on page 745) <i>REBIND</i> (on page 753)

**NAME**

TYPFMLNM — Data Type Family Name

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'0030'

**Length** \*

**Class** CLASS

**Sprcls** NAME - Name

Data Type Family Name (TYPFMLNM) specifies the name of a data type family. A type family is a set of data types designed to be used together in some processing environment.

See *TYPDEF* (on page 1025).

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'0030'	
<b>value</b>	INSTANCE_OF	NAMDR - Name Data
	ENUVAL	'QSQL'
	REQUIRED	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *TYPDEF* (on page 1025)

**Semantic** *TYPDEF* (on page 1025)

## NAME

TYPESQLDA — Type of SQL Descriptor Area

## DESCRIPTION (Semantic)

**Dictionary** QDDRDBD  
**Codepoint** X'2146'  
**Length** \*  
**Class** CLASS  
**Sprcls** STRING - String

Specifies the type of SQLDA descriptor area to return that applies to the SQL statement identified by the command. The SQL descriptor area is obtained by the target SQLAM. Refer to the SQLDTAGRP FD:OCA descriptor in the DRDA Reference for details on the different types of descriptor areas that can be returned.

Three types of SQL descriptor area can be requested for either input or output variable and column descriptions:

- Request a standard SQLDA
- Request a light SQLDA
- Request an extended SQLDA

TYPESQLDA determines the type of SQLDARD or SQLCINRD to be returned. Three types of SQLDA can be requested; a light, standard, or extended version. Each type provides a different set of descriptive information. The light SQLDA provides minimal descriptive information. The standard SQLDA provides the same descriptive information as was defined for SQLAM level 6 and earlier. The extended SQLDA provides additional descriptive information required by certain types of API, such as JDBC.

### Intermediate System Processing

The TYPESQLDA parameter, when defined for a given command, specifies both the type of the data to be described (input or output) and the amount of descriptor information to be returned (standard, extended, or light) with the command. All SQLAM levels that support the TYPESQLDA parameter support the return of standard descriptor information.

If the SQLAM manager level of the downstream server does not support the TYPESQLDA option for the command, then no TYPESQLDA parameter should be sent to the downstream server and no SQLDARD will be returned by the intermediate server to the upstream requester.

If the SQLAM manager level of the downstream server does not support the return of descriptor information for the type of data requested by the upstream requester, then neither the RTNSQLDA (if defined for the command) nor the TYPESQLDA parameter should be sent to the downstream server and no SQLDARD will be returned by the intermediate server to the upstream requester.

If the SQLAM manager level of the downstream server does not support the requested amount of descriptor information, then the TYPESQLDA parameter sent to the downstream server should specify the maximal SQLDARD information, according to the format specified in the DDM reference for the SQLAM level supported by the downstream server, for the requested type of data. The SQLDARD received from the downstream server may be sent to the upstream requester if it is preceded by a MGRLVLOVR object giving the SQLAM level value that describes the format and content of the SQLDARD.



<b>clsvar</b>	NIL	
<b>insvar</b>	INSTANCE VARIABLES	
length	*	
class	X'2146'	
value	INSTANCE_OF	BINDR - Binary Number Field QDDPRMD
	LENGTH	8
	OPTIONAL	
	ENUVAL	0
	NOTE	Standard output SQLDA.
	ENUVAL	1
	NOTE	Standard input SQLDA.
	ENUVAL	2
	NOTE	Light output SQLDA.
	ENUVAL	3
	NOTE	Light input SQLDA.
	ENUVAL	4
	NOTE	Extended output SQLDA.
	ENUVAL	5
	NOTE	Extended input SQLDA.
	DFTVAL	0
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

<b>insvar</b>	<i>DSCSQLSTT</i> (on page 304) <i>EXCSQLSTT</i> (on page 381) <i>OPNQRY</i> (on page 555) <i>PRPSQLSTT</i> (on page 636) <i>QDDPRMD</i> (on page 654)
<b>cmdrpy</b>	<i>DSCSQLSTT</i> (on page 304)
<b>Semantic</b>	<i>DSCSQLSTT</i> (on page 304) <i>EXCSQLSTT</i> (on page 381) <i>OPNQRY</i> (on page 555) <i>PRPSQLSTT</i> (on page 636) <i>SQLDARD</i> (on page 859) <i>SQLCINRD</i> (on page 857)

**NAME**

UNORDERED — Unordered

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'14B1'

**Length** \*

**Class** HELP

Unordered (UNORDERED) specifies that the parameters for DDM commands and reply messages may be in any sequence.

**SEE ALSO**

None.

**NAME**

UOWDSP — Unit of Work Disposition

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2115'

**Length** \*

**Class** CLASS

**Sprcls** SCALAR - Scalar Object

Unit of Work Disposition (UOWDSP) Scalar Object specifies the disposition of the last unit of work. If the disposition is committed, all of the updates in the unit of work are successfully applied. If the disposition is rolled back, all of the updates in the unit of work are removed.

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	5	
class	X'2115'	
value	INSTANCE_OF	HEXSTRDR - Hexadecimal String
	LENGTH	2
	ENUVAL	01
	NOTE	Means the unit of work was committed.
	ENUVAL	02
	NOTE	Means the unit of work was rolled back.
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *ENDUOWRM* (on page 343)

NAME

UOWID — Unit of Work Identifier

DESCRIPTION (Semantic)

**Dictionary** QDDBASD  
**Codepoint** X'11AA'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

Unit of Work Identifier (UOWID) uniquely identifies a unit of work within a network. UOWID is a name consisting of a unique network name, instance number, and commit sequence number. UOWID is generated at the source server or may be inherited from another source provided it can be mapped into the format of a UOWID.

Netname is a character string representation of a unique network address of the source server. In SNA environments netname is the fully qualified SNA network name blank padded on the right. The network identifier and LU name must be delimited by a “.”.

In TCP/IP environments, netname is the 8-byte hex character representation of the IP address, concatenated with a “.” which is concatenated with the 4-byte hex character representation of the TCP port number and blank padded on the right to 17 characters. The IP address and TCP port number must be delimited by a “.”.

A binary value instance, which could be a clock value, makes the combination of netname and instance unique for each application instance.

Sequence uniquely numbers each transaction that occurs during an instance of an application by starting at the value zero and incrementing by one on each commit or rollback.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	29	
class	X'11AA'	
instance	INSTANCE_OF LENGTH REQUIRED	BYTSTRDR - Byte String 6
netname	INSTANCE_OF LENGTH REQUIRED	CHRSTRDR - Character String Data Representation 17
sequence	INSTANCE_OF LENGTH REQUIRED	BINDR - Binary Number Field 16
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar**

*SYNCCTL* (on page 915)  
*SYNCRRD* (on page 981)  
*SYNCRSY* (on page 982)

**Semantic**

*CNNTKN* (on page 220)  
*SYNCPTOV* (on page 944)

NAME

UOWSTATE — Unit of Work State

DESCRIPTION (Semantic)

**Dictionary** QDDBASD  
**Codepoint** X'11AC'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

Unit of Work State (UOWSTATE) specifies the state of a unit of work.

Reset indicates that a SYNCPTMGR has no information about a unit of work. This may occur when the unit of work was rolled back or was committed and forgotten.

Committed indicates that the unit of work is in the commit state.

Unknown indicates that the outcome of the unit of work is unknown because the unit of work is still in progress and has not reached a commit decision at the resync server.

In-doubt indicates that the outcome of the unit of work is in-doubt because the unit of work has entered phase 1 of commit processing but the SYNCPTMGR has not been informed of the final outcome of the unit of work.

Cold indicates that a cold start was performed and the log needed for recovery is no longer accessible.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	5	
class	X'11AC'	
value	INSTANCE_OF	BYTDR - An 8-bit Data Representation Value
	ENUVAL	X'01' Reset
	ENUVAL	X'02' Committed
	ENUVAL	X'03' Unknown
	ENUVAL	X'04' In-doubt
	ENUVAL	X'05' Cold
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

SEE ALSO

**insvar** SYNCRRD (on page 981)  
 SYNCRSY (on page 982)

**NAME**

USADATFMT — USA Date Format

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'242A'**Length** \***Class** codpnt

USA Date Format (USADATFMT) specifies that dates in the SQL statements are in the USA date format:

mm/dd/yyyy

where / is a slash (with the Graphic Character Global Identifier (GCGID) SP12) character. For more information on GCGID, see *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar**                    *STTDATFMT* (on page 894)

**NAME**

USATIMFMT — USA Time Format

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'242F'**Length** \***Class** CODPNT

USA Time Format (USATIMFMT) specifies that times in the SQL statements are in the USA time format:

hh:mm xM  
x = A or P

where ":" is a colon (with the Graphic Character Global Identifier (GCGID) SP11) character, and A, M, and P are capital (with the Graphic Character Identifier (GCGID) LA02) characters. AM designates the time from midnight to noon; PM designates the time from noon to midnight. For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO****insvar** *STTIMFMT* (on page 900)



**NAME**

USRENCPWD — User ID and Password Encryption

**DESCRIPTION (Semantic)****Dictionary** QDDPRMD**Length** \***Class** CONSTANT

The User ID and Password Encryption Mechanism (USRENCPWD) specifies the use of the password encryption mechanism. The mechanism authenticates the user like the user ID and password mechanism, but the password is encrypted using 56-bit DES. The DES encryption seed is generated by exchanging connection keys in the SECTKN instance variable on the ACCSEC command and the ACCSEC reply data using the standard Diffie-Hellman key distribution algorithm to generate a shared private key. The application requester encrypts the password using the shared private key generated and the user ID. The user ID is passed in the USRID instance variable and the encrypted password is passed in the SECTKN instance variable on the SECCHK command. The application server decrypts the password using the shared private key and the user ID. The user ID and password are then passed to the local security user to be authenticated.

See the following terms for the specific meaning and function of each security mechanism component:

DHENC	Diffie-Hellman Security Mechanism (see <i>DHENC</i> (on page 280))
NWPWDSEC	New Password Security Mechanism (see <i>NWPWDSEC</i> (on page 535))
USRIDSEC	User Identification Security Mechanism (see <i>USRIDSEC</i> (on page 1048))
USRSECOVR	User ID Security Overview (see <i>USRSECOVR</i> (on page 1050))

---

value	7
-------	---

**SEE ALSO**

<b>insvar</b>	<i>ACCSEC</i> (on page 52) <i>ACCSECRD</i> (on page 58) <i>SECCHK</i> (on page 800) <i>SECMEC</i> (on page 811)
<b>Semantic</b>	<i>DHENC</i> (on page 280) <i>SECCHK</i> (on page 800) <i>SECMEC</i> (on page 811) <i>SECTKN</i> (on page 820)

**NAME**

USRID — User ID at the Target System

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'11A0'

**Length** \*

**Class** CLASS

**Sprcls** NAME - Name

User ID at the Target System (USRID) specifies the target-defined end-user name (identifier). The target server is responsible for any authentication and verification of the end-user name. DDM architecture does not define the semantics or syntax of USRIDs.

With certain SECMECs, the USRID can be encrypted.

The target server can optionally *fold* the user ID if it is not in the correct form for the target security manager. Folding is the process of modifying the characters in the user ID from one form to another. For instance, the original user ID may be in mixed case. The user ID could be folded to all uppercase or to all lowercase. Folding, if done on the original data, is not reversible.

The DDM architecture treats the USRID as a byte string. The SECMGR is aware of and has knowledge of the contents of the USRID. DDM does not know or care about the specific contents of this field. If the reader needs to know the actual contents, see *USRSECOVR* (on page 1050).

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'11A0'	
<b>value</b>	INSTANCE_OF	CHRSTRDR - Character String Data Representation
	MINLEN	1
	MAXLEN	255
	REQUIRED	
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

- insvar** ACCRDBRM (on page 48)  
SECCHK (on page 800)
- Semantic** DHENC (on page 280)  
PWDENC (on page 644)  
SECCHK (on page 800)  
SECOVR (on page 819)  
USRENCPWD (on page 1043)  
USRSECOVR (on page 1050)

**NAME**

USRIDNWPWD — User ID, Password, and New Password Security Mechanism

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Length** \*

**Class** CONSTANT

The User ID, Password, and New Password Security Mechanism (USRIDNWPWD) specifies a valid security mechanism combination for the ACCSEC command.

See the following terms for the specific meaning and function of each security mechanism:

NWPWDSEC New Password Security Mechanism (see *NWPWDSEC* (on page 535))

PWDSEC Password Security Mechanism (see *PWDSEC* (on page 649))

USRIDSEC User Identification Security Mechanism (see *USRIDSEC* (on page 1048))

USRSECOVR User ID Security Overview (see *USRSECOVR* (on page 1050))

---

value 5

**SEE ALSO**

**insvar** *SECCHK* (on page 800)  
*SECMEC* (on page 811)

**Semantic** *SECCHK* (on page 800)  
*SECMEC* (on page 811)  
*USRSECOVR* (on page 1050)

**NAME**

USRIDONL — User ID Only Security Mechanism

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Length** \*

**Class** CONSTANT

The User ID Only Security Mechanism (USRIDONL) specifies a valid security mechanism combination for the ACCSEC command.

See the following term for the specific meaning and function of the security mechanism:

USRIDSEC      User Identification Security Mechanism (see *USRIDSEC* (on page 1048))

---

value                      4

**SEE ALSO**

**insvar**                      *SECCHK* (on page 800)  
                                   *SECMEC* (on page 811)

**Semantic**                    *SECCHK* (on page 800)  
                                   *SECMEC* (on page 811)

**NAME**

USRIDPWD — User ID and Password Security Mechanism

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Length** \*

**Class** CONSTANT

The User ID and Password Security Mechanism (USRIDPWD) specifies a valid security mechanism combination for the ACCSEC command.

See the following terms for the specific meaning and function of each security mechanism:

PWDSEC Password Security Mechanism (see *PWDSEC* (on page 649))

USRIDSEC User Identification Security Mechanism (see *USRIDSEC* (on page 1048))

USRSECOVR User ID Security Overview (see *USRSECOVR* (on page 1050))

---

value 3

**SEE ALSO**

**insvar** *SECCHK* (on page 800)  
*SECMEC* (on page 811)

**Semantic** *SECCHK* (on page 800)  
*SECMEC* (on page 811)  
*USRSECOVR* (on page 1050)

**NAME**

USRIDSEC — User Identification Security Mechanism

**DESCRIPTION (Semantic)**

**Dictionary** QDDTTRD

**Length** \*

**Class** HELP

The User Identification Security Mechanism (USRIDSEC) specifies the use of the user identification security mechanism.

**SEE ALSO**

**Semantic**        *SECMEC* (on page 811)  
                      *USRENCPWD* (on page 1043)  
                      *USRIDNWPWD* (on page 1045)  
                      *USRIDONL* (on page 1046)  
                      *USRIDPWD* (on page 1047)  
                      *USRSBSPWD* (on page 1049)  
                      *USRSECOVR* (on page 1050)

**NAME**

USRSBSPWD — User ID and Password Substitute

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Length** \*

**Class** CONSTANT

The User ID and Password Substitute Mechanism (USRSBSPWD) specifies the use of the substitute password mechanism. The mechanism authenticates the user like the user ID and password mechanism, but the 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.

The SECTKN parameter is used to flow the application requester's encryption seed to the application server with the ACCSEC command, and SECTKN is also used with the ACCSECRD reply data to return the application server's encryption seed to the application requester.

See the following terms for the specific meaning and function of each security mechanism component:

- PWDSBS Password Substitute Security Mechanism (see *PWDSBS* (on page 646))
- PWDSEC Password Security Mechanism (see *PWDSEC* (on page 649))
- USRIDSEC User Identification Security Mechanism (see *USRIDSEC* (on page 1048))
- USRSECOVR User ID Security Overview (see *USRSECOVR* (on page 1050))

---

value 6

**SEE ALSO**

- insvar** *ACCSEC* (on page 52)  
*ACCSECRD* (on page 58)  
*SECCHK* (on page 800)  
*SECMEC* (on page 811)
- Semantic** *SECCHK* (on page 800)  
*SECMEC* (on page 811)  
*SECTKN* (on page 820)

**NAME**

USRSECOVR — User ID Security Overview

**DESCRIPTION (Semantic)**

**Dictionary** QDDTTRD

**Length** \*

**Class** HELP

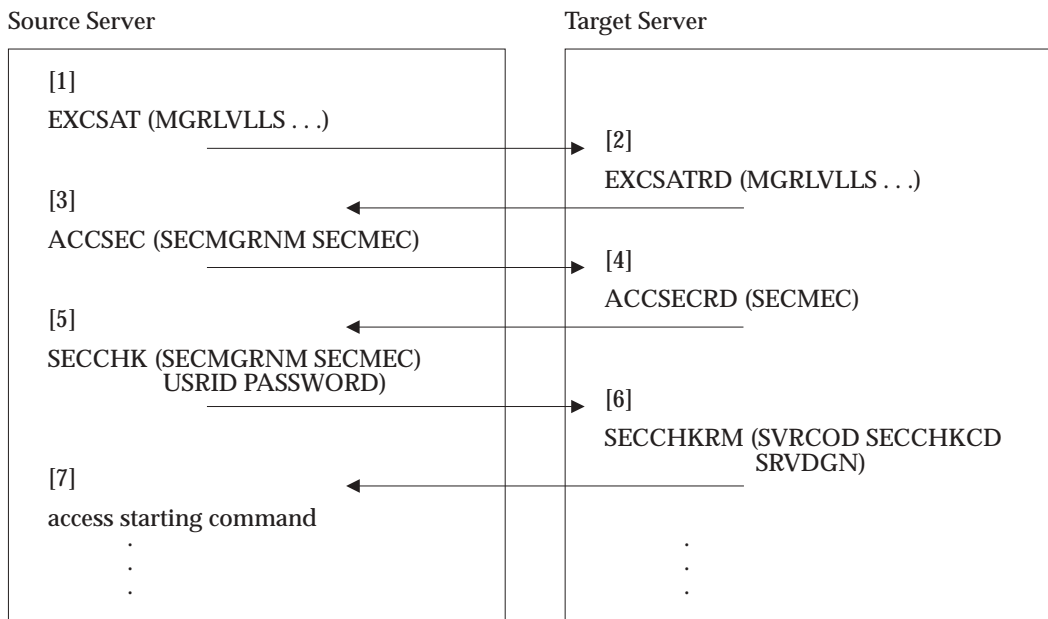
The User ID Security Overview (USRSECOVR) provides an overview of the User Identification (USRIDSEC) security mechanism component. USRIDSEC is a component of all of the SECMECs except the DCE security mechanism.

**User ID Security**

In some environments, the only security token required is the user ID to ensure proper authorization for accessing data in the target system. A source server acquires the user ID and sends it as part of the SECCHK command. The target server uses its security manager to identify and authenticate the end user to allow access to the target system's resources.

**User ID Security and Password Security**

Figure 3-120 indicates the DDM commands and replies that flow in the normal process of establishing a connection while using user ID and password security mechanisms.



**Figure 3-120 DDM USRID Security With a Password**

The following is a brief description of some of the parameters for the DDM commands. See the appropriate terms for a detailed description of the parameters.

1. The source server identifies the level of security it would like to operate at by identifying the SECMGR at manager Level 5 on the EXCSAT command.



```
EXCSAT (
  MGRLVLLS (
    MGRLVL (SECMGR, 5)
    .
    .
    .) )
```

- The target server receives the EXCSAT command and builds a reply data object with the SECMGR at manager Level 5 indicating it can operate at that security level. The target server sends the EXCSATRD reply data to the source server.

```
EXCSATRD (
  MGRLVLLS (
    MGRLVL (SECMGR, 5)
    .
    .
    .) )
```

- The source server receives the EXCSATRD reply data. The source server specifies the USRIDPWD security mechanism combination for authentication processing.

The SECMEC parameter indicates the security mechanism combination to use. This example assumes that the user ID and the password security mechanisms have been chosen.

- The target server receives the ACCSEC command. It supports the security mechanism combination identified in the SECMEC parameter. The target server returns the SECMEC parameter matching the source server in the ACCSECRD object.

If the target server does not support the security mechanism combination specified in the SECMEC parameter on the ACCSEC command, the target server returns the security mechanism combination values that it does support in the SECMEC parameter in the ACCSECRD object.

- The source server receives the ACCSECRD object and generates the security tokens required for security processing. The actual process to generate the security tokens (USRID and PASSWORD parameter values) is not specified by DDM. The source server may either generate the tokens, or it may call a security resource manager to generate the tokens.

The source server passes the tokens in the USRID and PASSWORD parameters on the SECCHK command.

- The target server receives the SECCHK command and uses the values in the received parameters to perform end-user authentication and other security checks.

The actual process to verify the security tokens (USRID and PASSWORD parameter values) is not specified by DDM. The target server may either process the values itself or it may call a security resource manager to process the values. Assuming the security mechanisms are the password and user ID, the inputs to the security verification processing are:

- The USRID parameter value received on SECCHK
- The PASSWORD parameter value received on SECCHK

The target server generates a SECCHKRM to return to the source server. The SECCHKCD parameter identifies the status of the security processing. A failure to authenticate the end user or to successfully pass the security processing results in the SVRCOD parameter value set to greater than INFO.

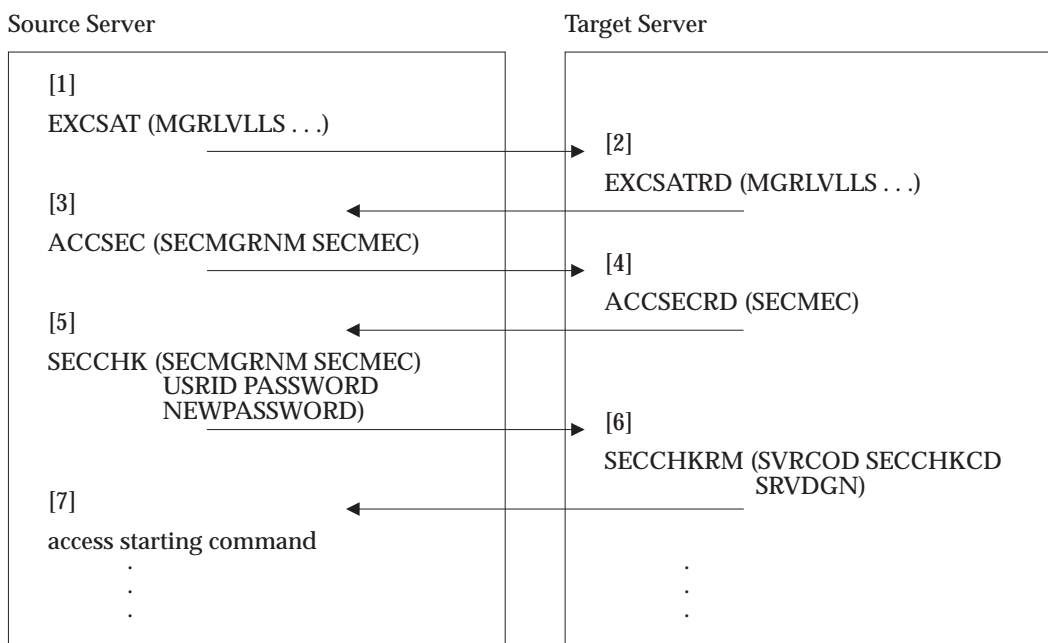
- The source server receives the SECCHKRM. Assuming security processing is successful, the source server sends any data access starting command to the target server.

If security processing fails, the source server might attempt recovery before returning to the application. If the error condition is not recoverable, the source server returns to the application with an error indicating a security verification failure.

If a source server is connecting to multiple target servers, the security information shared between each source server and target server pair is independent from each of the other pairs.

**User ID Security and New Password Security**

Figure 3-121 indicates the DDM commands and replies that flow in the normal process of establishing a connection while using the user ID and new password security mechanisms.



**Figure 3-121** DDM USRID Security with a New Password

The following is a brief description of some of the parameters for the DDM commands. See the appropriate terms for a detailed description of the parameters.

- The source server identifies the level of security it would like to operate at by identifying the SECMGR at manager Level 5 on the EXCSAT command.

```

EXCSAT (
    MGRLVLLS (
        MGRLVL (SECMGR, 5)
        .
        .
        .) )
    
```

- The target server receives the EXCSAT command and builds a reply data object with the SECMGR at manager Level 5 indicating it can operate at that security level. The target server sends the EXCSATRD reply data to the source server.

```

EXCSATRD (
    MGRLVLLS (
        MGRLVL (SECMGR, 5)
        .
        .
        .) )

```

3. The source server receives the EXCSATRD reply data. The source server specifies the USRIDNWPWD security mechanism combination for authentication processing.

The SECMEC parameter indicates the security mechanism combination to use. This example assumes the user ID and new password security mechanisms have been chosen.

4. The target server receives the ACCSEC command. It supports the security mechanism combination identified in the SECMEC parameter. The target server returns the SECMEC parameter matching the source server in the ACCSECRD object.

If the target server does not support the security mechanism combination specified in the SECMEC parameter on the ACCSEC command, the target server returns the security mechanism combination values that it does support in the SECMEC parameter in the ACCSECRD object.

5. The source server receives the ACCSECRD object and generates the security tokens required for security processing. The actual process to generate the security tokens (USRID, PASSWORD, and NEWPASSWORD parameter values) is not specified by DDM. The source server may either generate the tokens, or it may call a security resource manager to generate the tokens.

The source server passes the tokens in the USRID, PASSWORD, and NEWPASSWORD parameters on the SECCHK command.

6. The target server receives the SECCHK command and uses the values in the received parameters to perform end-user authentication and other security checks.

The actual process to verify the security tokens (USRID, PASSWORD, and NEWPASSWORD parameter values) is not specified by DDM. The target server may either process the values itself or it may call a security resource manager to process the values. Assuming the security mechanisms are the user ID and new password mechanisms, the inputs to security verification processing are:

- The USRID parameter value received on SECCHK
- The PASSWORD parameter value received on SECCHK
- The NEWPASSWORD parameter value received on SECCHK

The target server generates a SECCHKRM to return to the source server. The SECCHKCD parameter identifies the status of the security processing. A failure to authenticate the end user or to successfully pass the security processing results in the SVRCOD parameter value set to greater than INFO.

7. The source server receives the SECCHKRM. Assuming security processing is successful, the source server sends any data access starting command to the target server.

If security processing fails, the source server might attempt recovery before returning to the application. If the error condition is not recoverable, the source server returns to the application with an error indicating a security verification failure.

If a source server is connecting to multiple target servers, the security information shared between each source server and target server pair is independent from each of the other pairs.

### **User ID and Password Substitution Security**

One method of avoiding the sending of a password in the clear is to use a password substitute. The algorithm chosen is almost identical to that used in the SNA APPC architecture. A random seed is sent to the application server on ACCSEC in the SECTKN parameter, and a second random seed generated by the application server is returned on ACCSECRD. The seeds are used as described in the PWDSBS term to operate on the password at both sides and to validate the password at the application server, the substitute passwords generated by both the application requester and the application server are compared.

The flows for this mechanism are identical to those for user ID with password security with the exception that the seeds flow in SECTKN on ACCSEC and ACCSECRD, and that a SECCHKCD can flow back to the application requester with ACCSECRD if the SECTKN is missing on ACCSEC or if it is trivial (all binary zeros).

The strategic DRDA method of password encryption is DHENC described below.

### **User ID and Strong Password Substitution Security**

The strong password substitution security mechanism is identical to the original password substitute mechanism, except for the algorithm used to generate the password substitute, and the length of the password substitute (20 bytes compared with 8 for the original). The flows are the same. See the PWDSSB term for details of the algorithm.

### **User ID and Encrypted Password Security**

This method of encrypting the password that accompanies the user ID sent on a DRDA connection request differs from the substitute password mechanism in that the encrypted password is decrypted at the application server rather than used to compare with a password encrypted by the application server. The details of the encryption and decryption processes are described in the DDM DHENC term.

The flows for this mechanism are identical to those for user ID with password substitution security.

### **Encrypted User ID and Password Security**

One method of avoiding the sending of the user ID and password in the clear is to encrypt the user ID and password. The flows for this mechanism are identical to those for User ID Security and Password Security, except that keys flow in SECTKNs on the ACCSEC and ACCSECRD, and that a SECCHKCD can flow back to the application requester with ACCSECRD if the SECTKN is missing on ACCSEC or if it is trivial (all binary zeros). Further, the encrypted values of the user ID and password flow as two SECTKNs on the SECCHK command sent to the server.

The encrypted user ID and password are decrypted by the application server. The details of the encryption and decryption processes are described in the DHENC term.

### **Encrypted User ID Security and New Password Security**

One method of avoiding the sending of the user ID, password, and new password in the clear is to encrypt the user ID, password, and new password. The flows for this mechanism are identical to those for User ID Security and New Password Security, except that keys flow in SECTKNs on the ACCSEC and ACCSECRD, and that a SECCHKCD can flow back to the application requester with ACCSECRD if the SECTKN is missing on ACCSEC or if it is trivial (all binary zeros). Further, the encrypted values of the user ID, password, and new password flow as three SECTKNs on the SECCHK command sent to the server. The encrypted user ID, password, and new password are decrypted by the application server. The details of the encryption and

decryption processes are described in the DHENC term.

**SEE ALSO**

<b>insvar</b>	<i>ACCSEC</i> (on page 52) <i>ACCSECRD</i> (on page 58) <i>SECCHK</i> (on page 800)
<b>Semantic</b>	<i>DHENC</i> (on page 280) <i>EUSRIDNWPWD</i> (on page 357) <i>EUSRIDPWD</i> (on page 358) <i>NEWPASSWORD</i> (on page 531) <i>NWPWDSEC</i> (on page 535) <i>PASSWORD</i> (on page 578) <i>PWDSBS</i> (on page 646) <i>PWDSEC</i> (on page 649) <i>PWDSSB</i> (on page 650) <i>SECMGR</i> (on page 814) <i>SECOVR</i> (on page 819) <i>USRENCPWD</i> (on page 1043) <i>USRID</i> (on page 1044) <i>USRIDNWPWD</i> (on page 1045) <i>USRIDPWD</i> (on page 1047) <i>USRSBSPWD</i> (on page 1049) <i>USRSSBPWD</i> (on page 1056)

**NAME**

USRSSBPWD — User ID and Strong Password Substitute

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD  
**Length** \*  
**Class** CONSTANT

The Strong Password Substitution Security Mechanism (PWDSSB) specifies the use of a strong password substitute. See *USRSECOVR* (on page 1050) for more information about this mechanism.

The mechanism authenticates the user like the user ID and password mechanism, but the password does not flow. A password substitute is generated using the SHA-1 algorithm (see *PWDSSB* (on page 650)), and is sent to the application server. The application server generates a password substitute using the same algorithm and compares it with the application requester's password substitute. If equal, the user is authenticated.

The SECTKN parameter is used to flow the client and server encryption seeds on the ACCSEC and ACCSECRD commands.

See the following terms for the specific meaning and function of each security mechanism component:

- PWDSEC Password Security Mechanism (see *PWDSEC* (on page 649))
- PWDSSB Strong Password Substitute Security Mechanism (see *PWDSBS* (on page 646))
- USRIDSEC User Identification Security Mechanism (see *USRIDSEC* (on page 1048))
- USRSECOVR User ID Security Overview (see *USRSECOVR* (on page 1050))

---

value 8

**SEE ALSO**

- insvar** *ACCSEC* (on page 52)  
*ACCSECRD* (on page 58)  
*SECMEC* (on page 811)

**NAME**

VALNSPRM — Parameter Value Not Supported

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1252'

**Length** \*

**Class** CLASS

**Sprcls** RPYMSG - Reply Message

Parameter Value Not Supported (VALNSPRM) Reply Message indicates that the parameter value specified is either not recognized or not supported for the specified parameter.

The VALNSPRM can only be returned in accordance with the rules specified for DDM subsetting (see *SUBSETS* (on page 902)).

The codepoint of the command parameter in error is returned as a parameter in this message.

**DSS Carrier: RPYDSS**

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
<b>length</b>	*	
<b>class</b>	X'1252'	
<b>codpnt</b>	INSTANCE_OF REQUIRED NOTE	CODPNT - Codepoint  Return the codepoint of the parameter whose value is not supported.
<b>rdbnam</b>	INSTANCE_OF MINLVL OPTIONAL	RDBNAM - Relational Database Name 3
<b>reccnt</b>	INSTANCE_OF MINVAL OPTIONAL NOTE  NOTE  MINLVL	RECCNT - Record Count 0  Required for requests to insert multiple records in a file.  This parameter is not returned by commands that operate on RDBs.  3
<b>svrdgn</b>	INSTANCE_OF OPTIONAL	SRVDGN - Server Diagnostic Information
<b>svrcod</b>	INSTANCE_OF REQUIRED ENUVAL	SVRCOD - Severity Code  8 - ERROR - Error Severity Code
<b>clscmd</b>	NIL	

**inscmd**

NIL

**SEE ALSO****cmdrpy**

*ACCRDB* (on page 42)  
*ACCSEC* (on page 52)  
*BGNBND* (on page 110)  
*BNDSQLSTT* (on page 136)  
*CLSQRY* (on page 165)  
*CNTQRY* (on page 222)  
*DRPPKG* (on page 293)  
*DSCRDBTBL* (on page 300)  
*DSCSQLSTT* (on page 304)  
*ENDATMCHN* (on page 333)  
*ENDBND* (on page 336)  
*EXCSAT* (on page 363)  
*EXCSQLIMM* (on page 371)  
*EXCSQLSET* (on page 377)  
*EXCSQLSTT* (on page 381)  
*INTRDBRQS* (on page 445)  
*OPNQRY* (on page 555)  
*PRPSQLSTT* (on page 636)  
*RDBCMM* (on page 728)  
*RDBRLLBCK* (on page 745)  
*REBIND* (on page 753)  
*SECCHK* (on page 800)  
*SYNCCTL* (on page 915)  
*SYNCRSY* (on page 982)

**Semantic**

*ACCRDB* (on page 42)  
*SUBSETS* (on page 902)



**NAME**

VRSNAM — Version Name

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1144'

**Length** \*

**Class** CLASS

**Sprcls** NAME - Name

Version Name (VRSNAM) specifies the version name associated with an entity. The format of the version name is not architected.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
length	*	
class	X'1144'	
value	INSTANCE_OF	NAMDR - Name Data
	MAXLEN	254
	DFTVAL	''
	NOTE	The default value means "null".
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

**SEE ALSO**

**insvar** *BGNBND* (on page 110)  
*BGNBNDRM* (on page 117)  
*DRPPKG* (on page 293)  
*PKGRPLVRS* (on page 605)  
*REBIND* (on page 753)

**Semantic** *ENDBND* (on page 336)  
*PKGATHOPT* (on page 580)  
*PKGID* (on page 591)  
*PKGNAME* (on page 594)  
*PKGRPLALW* (on page 602)  
*PKGRPLNA* (on page 603)  
*PKGRPLOPT* (on page 604)  
*REBIND* (on page 753)  
*SQL* (on page 835)

**NAME**

WARNING — Warning Severity Code

**DESCRIPTION (Semantic)**

**Dictionary** QDDPRMD

**Codepoint** X'003D'

**Length** \*

**Class** CONSTANT

Warning Severity Code (WARNING) specifies that a reply message is a warning of a potential problem in the processing of a command.

Further processing of a command depends on the architected specifications of the specific command, the error condition, and the environment in which it is being executed.

---

value 4

**SEE ALSO**

**insvar**            *ACCRDBRM* (on page 48)  
                       *CMDCHKRM* (on page 173)  
                       *CMDNSPRM* (on page 176)  
                       *ENDQRYRM* (on page 342)  
                       *ENDUOWRM* (on page 343)  
                       *QRYNOPRM* (on page 690)  
                       *RSCLMTRM* (on page 778)  
                       *SVRCOD* (on page 911)

**Semantic**        *ACCRDB* (on page 42)  
                       *ACCRDBRM* (on page 48)  
                       *DCESECOVR* (on page 253)  
                       *EDTASECOVR* (on page 323)  
                       *PLGINOVR* (on page 615)

**NAME**

XAFLAGS - XA Flags

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1903'

**Length** \*

**Class** CLASS

**Sprcls** FIELD - A Discrete Unit of Work

Flags used on an XAmgr protected connection to convey information to the target server. Multiple values can be combined by having their contents OR'ed.

**TMFAIL** Dissociate transaction branch and mark transaction branch as Rollback only.

**TMJOIN** Joining existing transaction branch.

**TMLCS** Require Loosely Coupled Support; that is, share RDB resources with all transaction branches with the same GTRID.

**TMLOCAL** Local Transaction.

**TMNOFLAGS** No features selected.

**TMONEPHASE** Use one-phase commit optimization.

**TMRECOVER** In Recovery mode.

**TMRESUME** Resuming association with suspended transaction branch.

**TMSUCCESS** Dissociate Transaction branch for XAMGR protected connection.

**TMSUSPEND** Suspending transaction branch, not ending it.

See *XAMGROV* (on page 1066) for details of which combinations of XAFLAGS can be used with the various SYNCTYPES.

<b>clsvar</b>	NIL	
<b>insvar</b>	INSTANCE VARIABLES	
<b>length</b>	8	
<b>class</b>	X'1903'	
<b>value</b>	INSTANCE_OF	HEXSTRDR
	REQUIRED	
	LENGTH	8
	ENUVAL	X'00000000' TMNOFLAGS
	ENUVAL	X'00080000' TMRECOVER
	ENUVAL	X'00200000' TMJOIN
	ENUVAL	X'00800000' TMLCS
	ENUVAL	X'02000000' TMSUSPEND
	ENUVAL	X'04000000' TMSUCCESS
	ENUVAL	X'08000000' TMRESUME
	ENUVAL	X'20000000' TMFAIL

	ENUVAL	X'40000000' TMONEPHASE
	DFTVAL	X'00000000' TMNOFLAGS
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

- insvar**            *SYNCCTL* (on page 915)
- Semantic**        *AGENT* (on page 61)
- SYNCCTL* (on page 915)
- SYNCTYPE* (on page 984)
- XAMGROV* (on page 1066)

**NAME**

XAMGR — XA Manager

**DESCRIPTION (Semantic)****Dictionary** QDDBASD**Codepoint** X'1C01'**Length** \***Class** CLASS**Sprcls** MANAGER - Resource Manager

The XA Manager (XAMGR) 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 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)
 

End 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)
 

Obtain a list of Prepared and Heuristically completed Transactions at the application server.
7. SYNCCTL(Forget)
 

Ask 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 in protecting a resource to the application server. The XAMGR on the application server conveys using enhanced DDM SYNCCR objects the response to the application requester. The connection is always protected by a presumed rollback protocol in case of network failures or general errors.

Transactions are associated with an XID. A valid XID represents a Global Transaction with one or branches, while a NULL XID represents a Local Transaction. The application is responsible for protecting all Global Transactions using two-phase protocol, but must drive local commit or rollback for Local Transactions.

In addition to supporting the DDM flow, the XAMGR in conjunction with the RDB, on the Application Server is required to:

1. Log and track all transaction branches
2. Implicitly provide resource sharing capabilities for tightly coupled and loosely coupled connections

**Manager-Level Compatibility**

The following table illustrates the XAMGR dependencies on other managers.

<b>DDM Levels</b>	<b>7</b>
XAMGR	7
<i>Manager Dependencies</i>	
CMNAPPC	3
CMNTCPIP	5

XAMGR level 7 uses enhanced DDM Syncctl and Syncprd objects to provide an XA Interface in DRDA, which is only valid when using the services of the XAMGR. XAMGR can be used with SNA LU 6.2 SYNC\_LEVEL(NONE) or TCP/IP. See XAMGROV (on page 1066).

<b>clsvar</b>	<b>NIL</b>
<b>insvar</b>	<b>NIL</b>
<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>
<b>mgrlvl</b>	<b>7</b>
<b>mgrdepls</b>	<b>MANAGER DEPENDENCY LIST</b>
X'1444'	INSTANCE_OF CMNAPPC - LU 6.2 Conversational Communications Manager MGRVLN 3 NOTE The communications manager must support SNA LU 6.2 SYNC_LEVEL(NONE).
X'1474'	INSTANCE_OF CMNTCPIP - TCP/IP Communication Manager MGRVLN 5
X'1403'	INSTANCE_OF AGENT - Agent MGRVLN 5
<b>vldattls</b>	<b>VALID ATTRIBUTES</b>
X'0019'	INSTANCE_OF HELP - Help Text
X'1452'	INSTANCE_OF MGRNAM - Manager Name
X'0045'	INSTANCE_OF TITLE - Title

**SEE ALSO**

- insvar**      *SYNCCRD* (on page 913)  
                 *SYNCCTL* (on page 915)
- Semantic**    *AGENT* (on page 61)  
                 *RDB* (on page 718)  
                 *RDBOVR* (on page 740)  
                 *SUBSETS* (on page 902)  
                 *SYNCCRD* (on page 913)  
                 *SYNCCTL* (on page 915)

*SYNCTYPE* (on page 984)  
*XAFLAGS* (on page 1061)  
*XAMGROV* (on page 1066)

## NAME

XAMGROV — Overview for XA Flows

## DESCRIPTION (Semantic)

**Dictionary** QDDBASD**Length** \***Class** HELP**Overview of The Open Group XA Protocol**

The following section briefly outlines the XA interface. It is expected that the reader is familiar with *DTP: The XA+ Specification*.

The Open Group has defined a *Distributed Transaction Processing (DTP) Reference Model* consisting of three elements:

1. Application Program (AP)
2. Transaction Manager (TM)
3. Resource Managers (RMs)

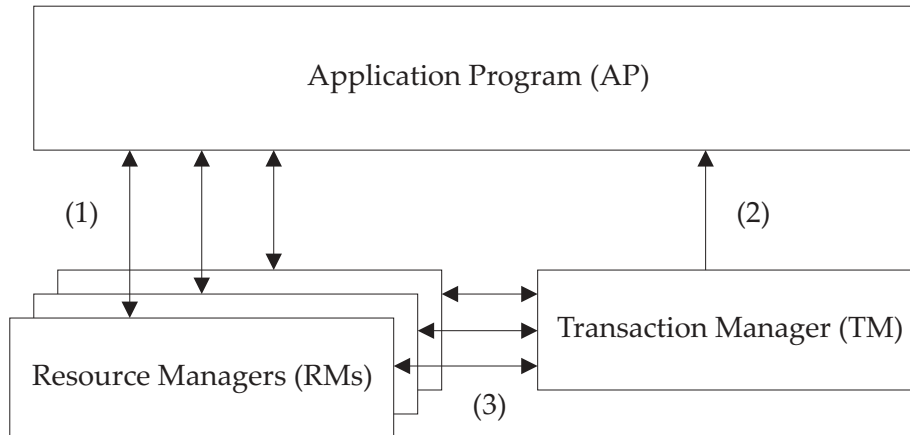
The figure below illustrates a local instance of a DTP system where the AP calls a TM to structure transactions. The boxes indicate software components in the DTP model. The arrows indicate the directions in control flows. There may be several DTP systems coexisting on the same processor. The boxes in the figure below are not necessarily separate processes, nor necessarily a single thread of control. Furthermore, the components of this model do not have invariable roles. For example, an RM might use the TX interface to do work in support of a transaction.

Interface (1) represents the interface an RM provides to an AP in order to allow the AP to use RM resources. Interface (2) allows an AP to define transaction boundaries through the interface provided by the TM. Interface (3) allows RMs and TMs to interact to exchange transaction information. Interface (3) is known as the XA interface. The XA interface is defined in the The Open Group *DTP: The XA Specification*. Following is a summary description for reference.

In order to make the diagram below more meaningful, an example implementation of the DTP system would be:

1. AP is an SQL application. Therefore the interface used in (1) is SQL.
2. TM is any software component or product which provides an XA-compliant transaction manager. Therefore the interface used in (2) is the interface the TM provides to the application. Interface (3) is the XA Interface.
3. RMs are typically DBMS. Interface (3) is the XA Interface and interface (1) is SQL.





The XA Transaction Manager is responsible for providing the following XA APIs for use by the RM:

- ax\_reg* Dynamically register a RM with a TM.
- ax\_unreg* Dynamically unregister an RM with a TM.

The Resource Manager is responsible for providing the following XA APIs for use by the TM:

- xa\_close* Close an RM. There is no DDM mapping for this API.
- xa\_commit* Commit a transaction.
- xa\_complete* Test an asynchronous XA operation for completion. There is no DDM mapping for this API.
- xa\_end* Dissociate the thread of control from a transaction branch.
- xa\_forget* Permit the RM to discard its knowledge of a heuristically-completed transaction branch.
- xa\_open* Initialize an RM for use by an AP.
- xa\_prepare* Ask the RM to prepare to commit a transaction branch.
- xa\_recover* Get a list of XIDs the RM has prepared or heuristically completed.
- xa\_rollback* Tell the RM to roll back a transaction branch.
- xa\_start* Start or resume a transaction branch—associate an XID with future work that the thread requests of the RM.

**Note:** DRDA provides functionality that allows a TM application to perform the operations required in protecting a DRDA resource. This architecture is only modeled after the XA protocol. However, a sample is provided showing how a Resource Manager can use this DRDA Transactional Processing support in providing an XA Interface to an XA-complaint Transactional Manager.

### DDM Transactional Processing Support

XAMGR level 7 is a manager object of DDM that provides a connection architecture that will allow the application to perform the operations involved in protecting a DRDA resource. The DRDA functionality is summarized below:

- SYNCCTL(New Unit of Work)

Provides the application with the ability to register/start a transaction with the DBMS and associate the connection with the transaction's XID. A valid XID represents a Global Transaction that requires two-phase protection, while a NULL XID represents an unprotected transaction. The functionality also provides the application with the ability to join or resume existing transaction branches at the application server. A timeout value is also specified, that allows the transaction to be rolled back when it exceeds the timeout limit.

- SYNCCTL(End association)

An application uses this functionality to inform the application server that it is ending a Global Transaction and is dissociating the XID from the connection. It also provides the application with the ability to suspend or fail a transaction branch. Suspend with migrate gives the application the capability of moving the resources associated with a transaction branch from one connection to another.

- SYNCCTL(Prepare to Commit)

Provides the application with the ability to perform the first phase of the two-phase protocol, which involves informing a DBMS to prepare the transaction branch for commit.

- SYNCCTL(Commit)

Provides the application with the ability to carry out the second phase of the two-phase protocol, which involves committing the Global Transaction. The function also allows the application to perform a one-phase optimization.

- SYNCCTL(Rollback)

Provides the application with the ability to roll back a Global Transaction. The function also allows the application to perform a one-phase optimization.

- SYNCCTL(Return Indoubt List)

At any given time, the application can use this function to obtain a list of Prepared and Heuristically completed Transactions at the DBMS. The application can then compare the list with its own and commit and roll back the transaction branches accordingly.

- SYNCCTL(Forget)

A DBMS will keep the knowledge of a heuristically completed transaction branch until it is authorized by the application to forget the transaction branch. The application uses this DRDA function to inform the DBMS which heuristically completed transaction branch it should forget.

Following are the DDM enhancements to the SYNCCTL and SYNCCRD objects. These enhancements are only valid when using the XAMGR.

**SYNCCTL DDM Object**

- Synctype X'0B', to end association of the transaction branch from the XA protected connection
- Synctype X'0C', to obtain a list of prepare and heuristically completed transaction branches
- Use of XID as the sole transaction identifier
- Parameters XIDSHR and FORGET no longer valid

**SYNCCRD DDM Object**

- New parameter XARETVAl used to inform the application requester the result of the XA operation
- Synctype no longer valid on the object
- A PRPHRCLST parameter used to return a list of prepared or heuristically completed transaction branches to the application requester

The XAMGR also provides the application with the ability to migrate cursors, RDB protected resources (for example, temp tables), and external savepoints from one XA protected connection to another. In addition, all XA protected connections can be pooled to different applications using the RLSCONV parameter. This pooling can occur as long as the connection is not associated with an XID.

**Transaction Identifier (XID)**

All transactions on an XA 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 the XA XID. The XID has two portions, the GTRID and the BQUAL. The XID format as defined by the The Open Group is shown below:

```
#define XIDDATASIZE 128          size in bytes
struct xid_t {
    long formatID;              format identifier
    long gtrid_length;          value 1-64
    long bqual_length;          value 1-64
    char data[XIDDATASIZE];    gtrid | bqual
};
typedef struct xid_t XID;
```

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. The RDB must optimize shared resources and locks to prevent any deadlocks from occurring between connections involved in the same Global Transaction but with different branches.

An application may want to perform an unprotected transaction on an XA protected connection. Such transactions are known as Local Transactions, and require no ending, two-phase coordination, or recovery.

**Note:** Local Transactions are not distributed. But the application must inform the application server that the transaction that is about to begin requires no protection. The application does this by registering the transaction with a NULL XID. A formatID of -1 represents a NULL XID. Once done with the transaction, the application must drive a Local commit or rollback thorough the RDB manager (that is, RDBCMM/RDBRLLBCK).

### **Tightly and Loosely-Coupled Transactions**

All work performed on an XA protected connection is associated with a transaction branch. The application begins the association and also ends the association with the transaction branch. The association and disassociation are done via the SYNCCTL(New UOW) and SYNCCTL(End) operations.

XA connections that are performing work that is to be associated with the same transaction branch are in a tightly-coupled relationship; that is, they are involved in the same Global Transaction and have the same branch qualifier. Regarding the DBMS isolation policies, these XA connections should be treated as one single entity, and all resources shared between these connections so as to prevent any deadlocks from occurring. On the other hand, XA connections that are involved in the same Global Transaction that have different transaction branches may be treated, with regard to the DBMS isolation policies, as separate Global Transactions, even though their work will be completed atomically. The XAFLAG (TMLCS) allows the application requester to inform the RDB when it requires resources and locks to be shared in order to prevent any deadlocks from occurring between these loosely-coupled connections.

### **Heuristically Completed Branches**

A DBMS may employ a heuristic decision in committing or rolling back a transaction branch independently of the application. This could result in shared resources being released and leaving the Global Transaction in an inconsistent state; for example, RDB initiated rollbacks or exceeding the timeout limit.

When the application ultimately directs the application server to complete the branch, the application server must report to the application that the branch has been heuristically completed, even if the decision matches the application's request. The application server must keep the knowledge of the transaction branch around, until the application authorizes it to forget about the transaction branch.

### **Failure and Recovery**

The application is responsible for initiating recovery with any given DBMS. The application maintains its own logs in order to drive resynchronization. The application requests a list of prepared or heuristically completed transaction branches from the application server. The application server returns a list of transaction identifiers (XIDs) which are currently prepared or have been heuristically completed. The application compares this list with its own list and commits or aborts relevant transaction branches.

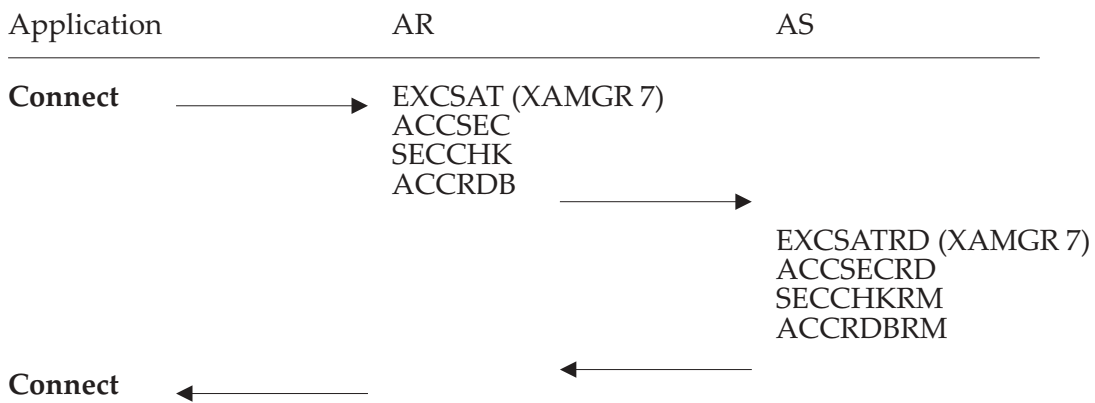
**DDM Flows Representing the XAMGR Functionality**

Following are samples showing the usage of the following XAMGR functionality:

1. SYNCCTL(New Unit of work)
  - Registering/Starting a Transaction with the DBMS and associating the connection with the transaction's XID.
2. SYNCCTL(End association)
  - End a Transaction with the DBMS and dissociating the connection from the transaction's XID. SYNCCTL(Prepare to commit)
    - Requesting the application server to Prepare a Transaction for the Commit phase.
3. SYNCCTL(Commit)
  - Committing the Transaction.
4. SYNCCTL(Rollback)
  - Rolling back the Transaction.
5. SYNCCTL(Return Indoubt List)
  - Obtain a list of Prepared and Heuristically completed Transactions at the application server.
6. SYNCCTL(Forget)
  - Ask the application server to forget about a heuristically completed transaction.

**Requesting XAMGR Support**

Following in a connect flow illustrating the request of XAMGR Level 7 support. In the sample below, the application server does support XAMGR and hence returns the EXCSATRD containing XAMGR. If the application server does not support the XAMGR, it would return XAMGR with Level 0, and the connection formed would be a DUOW unprotected.

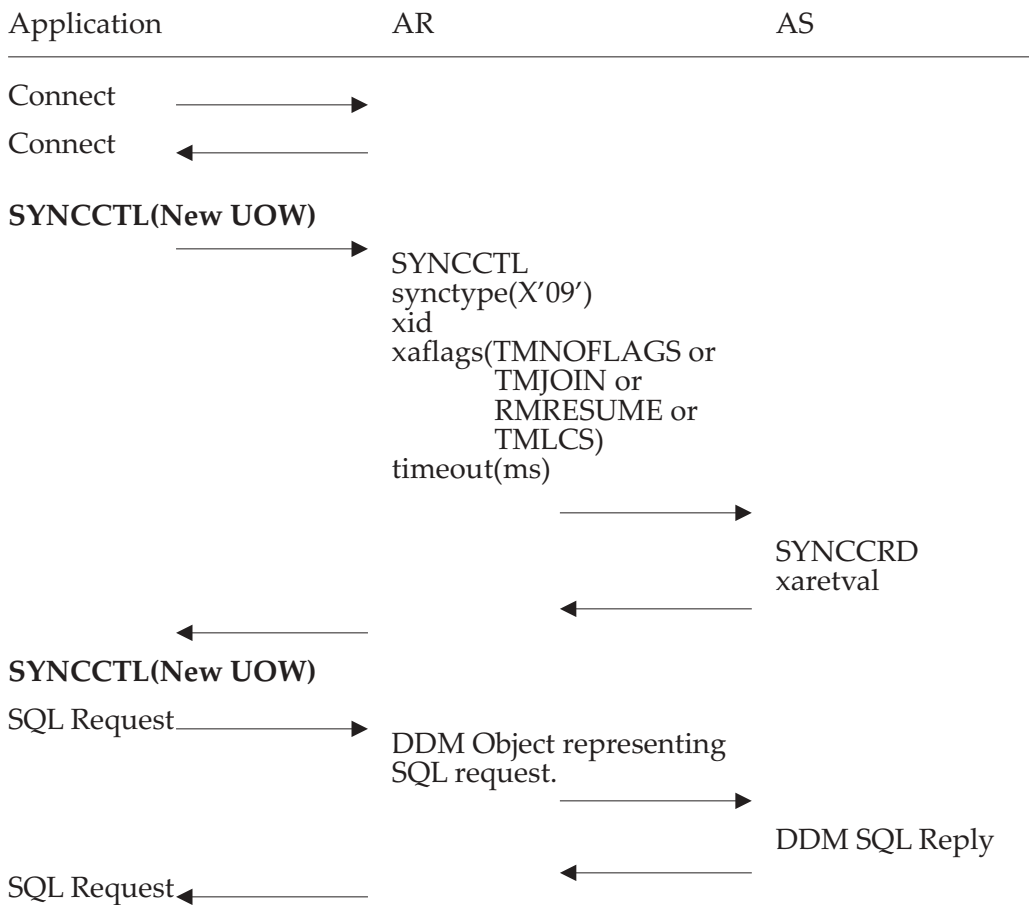


**Figure 3-122** Requesting XAMGR Support

This flow shows all objects are chained. It is not necessary to chain them.

**Starting a Transaction (SYNCCTL- New UOW)**

Once the connection has been formed, the application can begin doing work. But before doing so the transaction that is about to begin must be registered/started with the DBMS. Registration is the process by which the application provides the transaction identify (XID) to the AS. The XID represents the transaction or unit of work that is about to occur on this connection. The XID is logged by DBMS, so that work done under this XID can be committed or recovered at a later time.



**Figure 3-123** SYNCCTL(New UOW) Functionality Flow

This sample shows the SYNCCTL being sent before returning to the application. An application may elect to return immediately and chain the SYNCCTL to the first DDM object of the transaction. The timeout parameter indicates the number of milliseconds after which the RDB can timeout and roll back the transaction branch. This parameter is valid for Global Transactions only.

Once the reply has been returned and is successful, the XA protected connection can be associated with the transaction branch (XID) flown on the SYNCCTL(New UOW) request. That is, all work that will be done on this connection will be identified by this transaction branch. The XA connection is disassociated from the transaction branch by a SYNCCTL(End) request. No work can be issued on the connection until another SYNCCTL(New UOW) request is sent. The application could elect to associate a different XID to the XA connection.

Following are the valid XARETVALS that can be returned, when replying to a Register request:

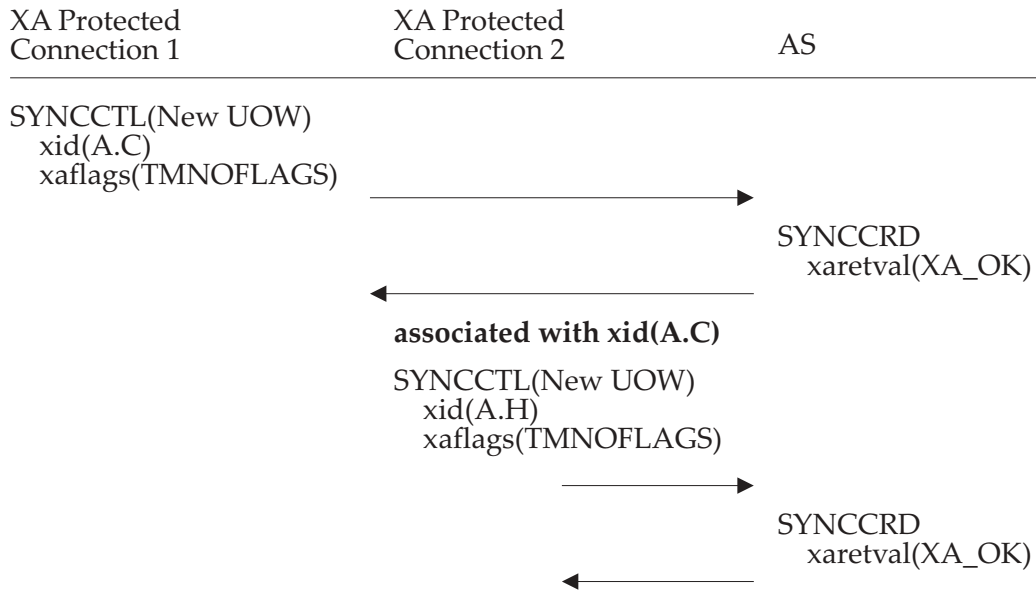
XA_OK	Normal Execution.
XA_LCSNOTSUPP	Sharing locks and resources is not supporting. Not a transaction error, just a warning.
XAER_RMERR	An error occurred associating the transaction branch with the XA protected connection.
XAER_RMFAIL	An error occurred that makes the XA protected connection unavailable.
XAER_PROTO	A protocol error has occurred.
XAER_OUTSIDE	The XA protected connection is doing work outside any Global Transaction.
XAER_NOTA	The XID is not known by the DBMS.
XAER_DUPID	The transaction branch already exists. The transaction branch should not be associated with the connection.
XAER_INVAL	Invalid XAFLAGS were specified.

The returning of the XA\_RB\* will result in the transaction branch being marked as rollback-only, and the connection dissociated from the transaction branch.

XA_RBROLLBACK	The DBMS has marked the transaction branch as rollback-only for an unspecified reason.
XA_RBCOMMFAIL	A communication failure has occurred.
XA_RBDEADLOCK	The DBMS detected a deadlock.
XA_RBINTEGRITY	The DBMS detected a violation of the integrity of its resources.

Following are the XAFLAGS that are valid on an SYNCCTL(New UOW):

TMNOFLAGS	Indicates that the XID contains a new branch qualifier (transaction branch). The application requires that all work about to begin on this connection be associated with this transaction branch. The figure below shows an example of starting a Global Transaction on an XA protected connection.
-----------	---



**Figure 3-124** Global Transactions (Tightly-Coupled)

TMLCS

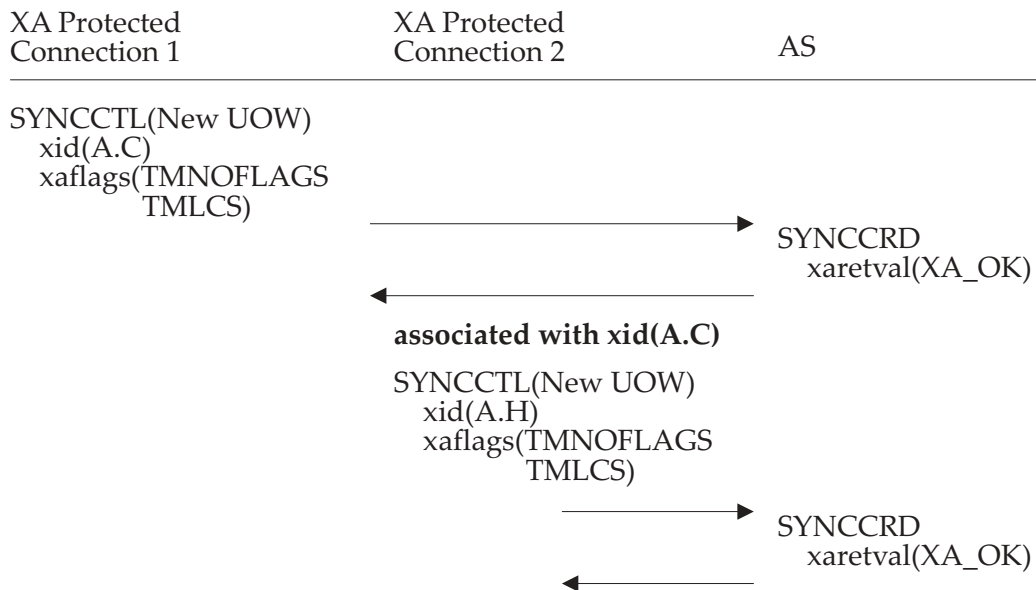
In the following example, both transaction A.C and A.H will be tightly related to each other; that is, A.C will lock, timeout, or deadlock if it attempts to access resources held by A.H. In order to make both transactions loosely related to each other, use the flag TMLCS at start time.

**Note:** This flag is only valid in conjunction with TMNOFLAGS.

A.C should be able to access resources held by A.H.

**Note:** If the RDB is unable to handle TMLCS, than it should return the value XA\_LCSNOTSUPP, but should continue to register the transaction. XA\_LCSNOTSUPP is a warning message that the RDB cannot share locks and resources and will be using its default lock mechanism (that is, will treat each XID as a unique transaction).



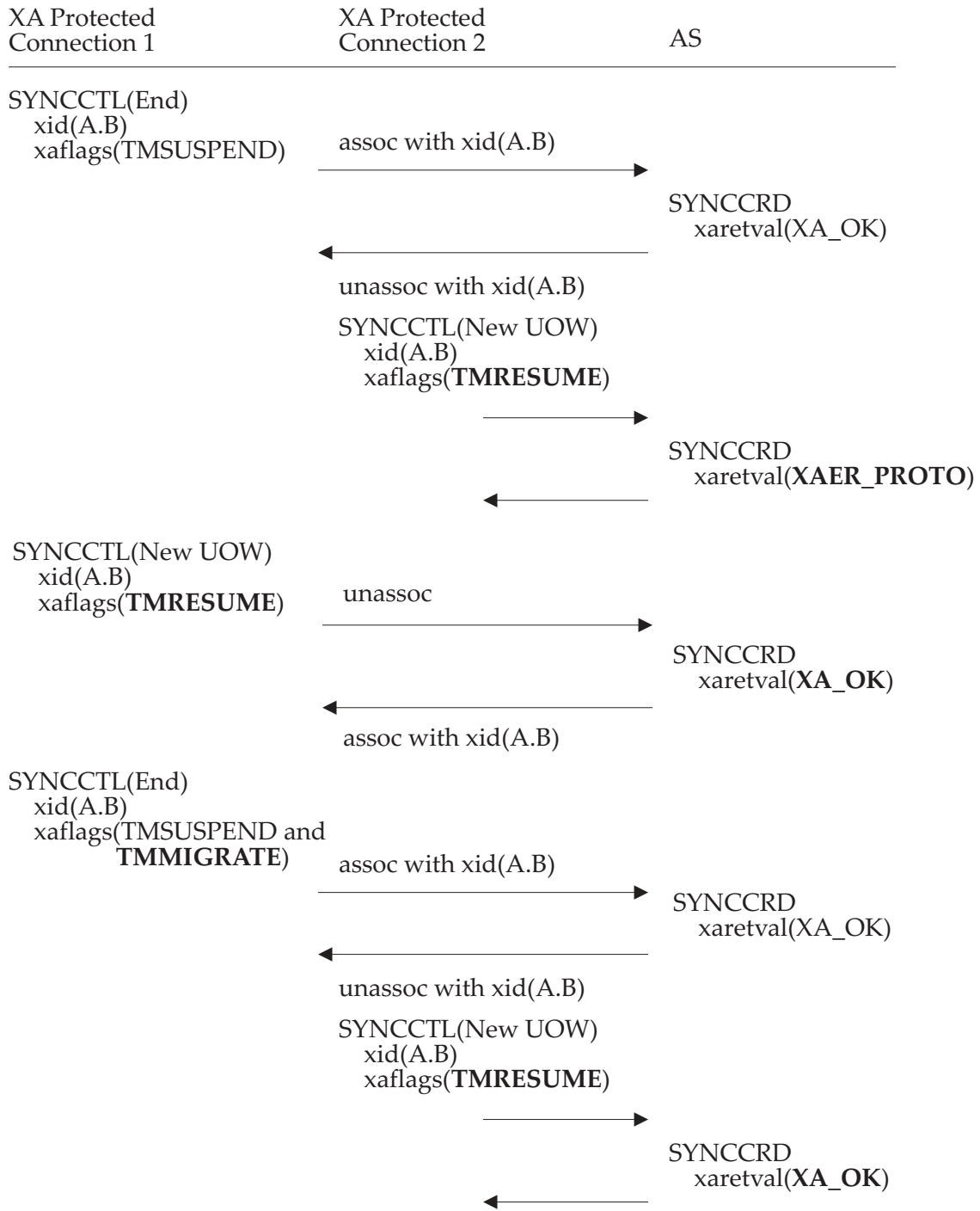


**Figure 3-125** Global Transactions (Loosely-Coupled)

TMRESUME

The XID represents a transaction branch that was suspended by this connection. The application wants to RESUME this transaction branch. A connection can only resume a transaction branch if it was the originator of the SUSPEND, or if the SUSPEND was marked for migration and the application requires that this connection should resume the suspended branch.

Both connections in the sample below are in a tightly-coupled relationship, and hence the RDB must share resource and locks so that no deadlocks will occur between the two connections.



**Figure 3-126** Resuming Suspended Transaction Branches

**TMJOIN**

The XID represents an existing transaction branch at the application server which has not yet been committed. The transaction is currently not associated with a connection or was suspended by another connection. The application wants this connection to JOIN this existing branch and include the work that is about to begin on the XA protected connection

with that existing branch. A connection can JOIN any existing branch that was SYNCCTL(End) with TMSUCCESS on any connection including itself, or a branch that was suspended without migrate, by a connection other than itself. See the figure below.

**Note:** In case of failure, all work that was previously and is currently associated with this transaction branch will be rolled back. Because this is a used branch qualifier, this XA protected connection has a tightly-coupled relationship with other XA protected connections that were previously associated with this transaction branch. The DBMS must guarantee that these XA protected connections are treated as a single entity with respect to its isolation policies and that resources are shared to prevent any deadlocks from occurring. The figure below shows an example of TMJOIN.

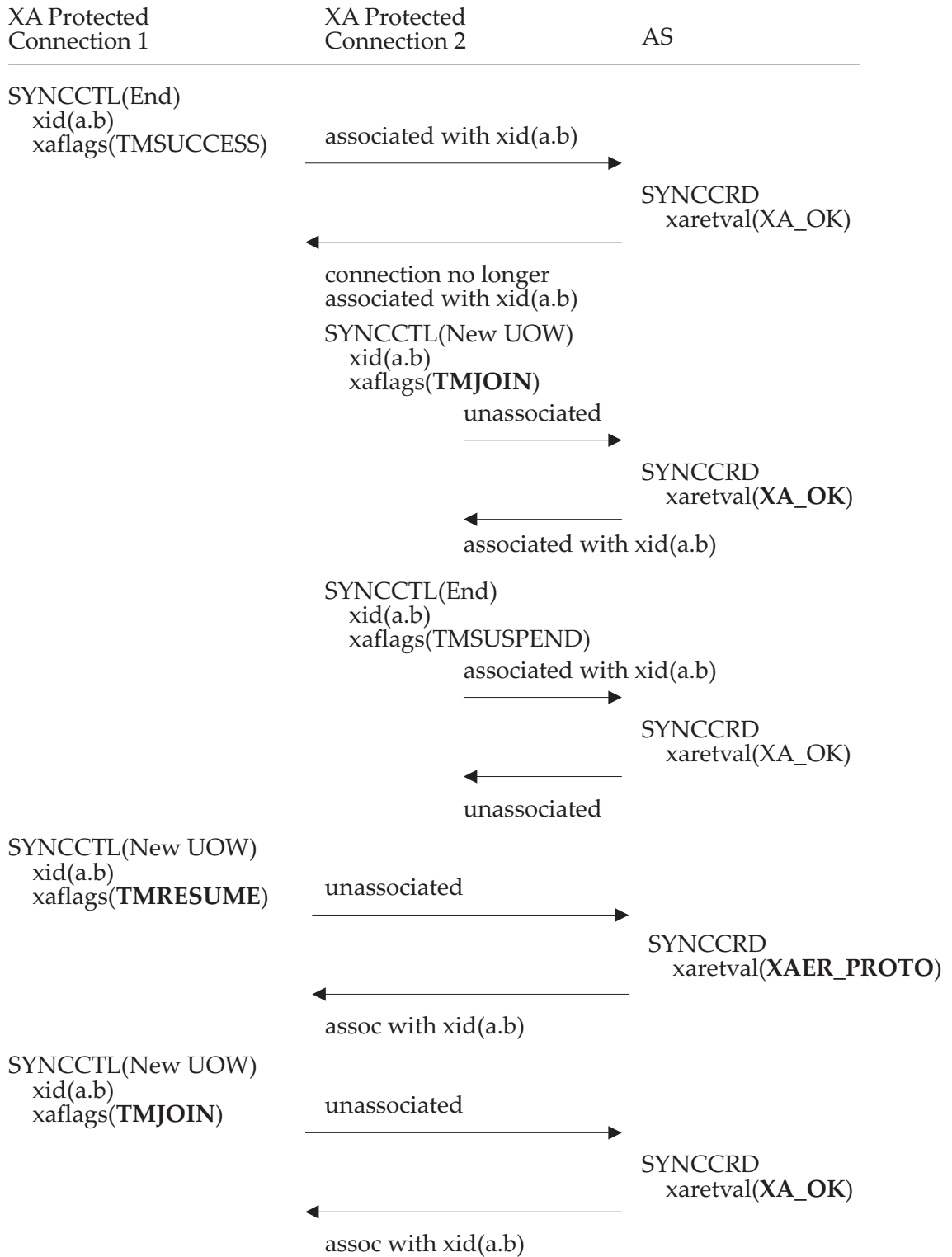
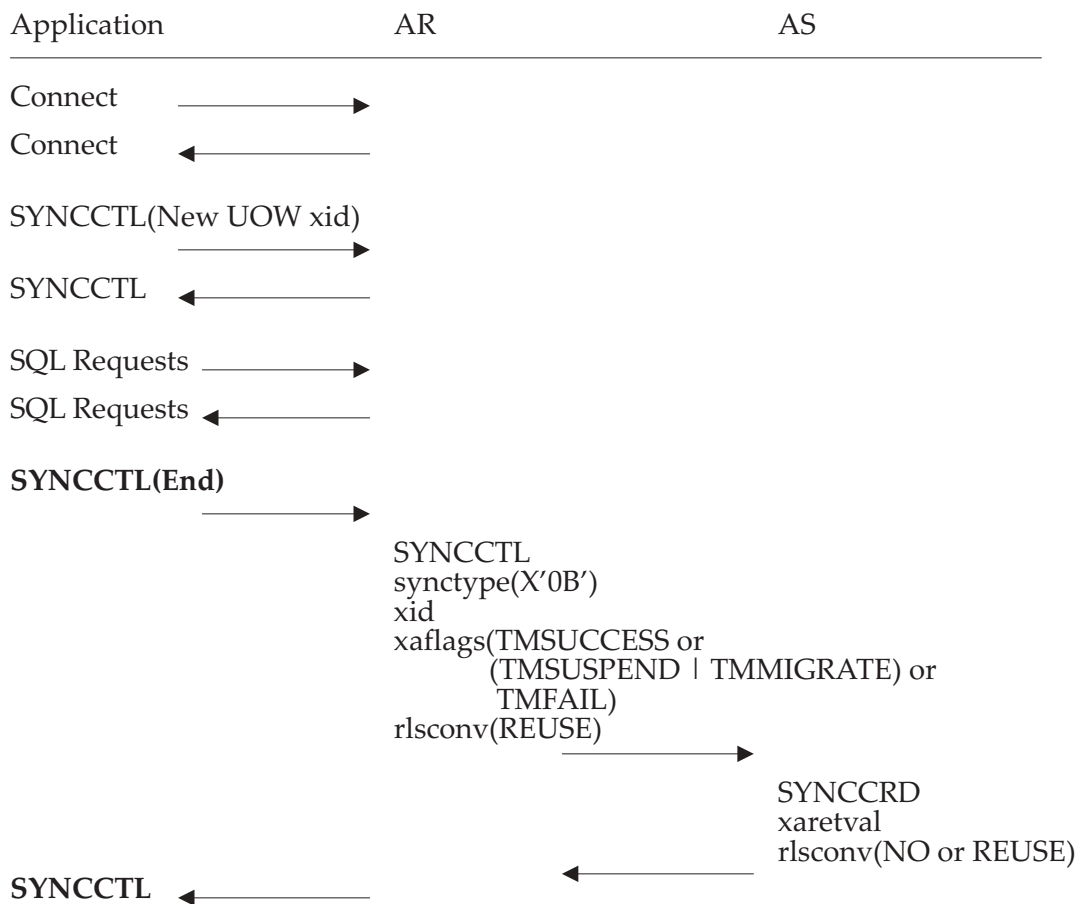


Figure 3-127 Example of TMJOIN Usage

**Ending a Transaction (SYNCCTL - End)**

The application must indicate the completion of a Global Transaction. It does this by sending a SYNCCTL('0B') to inform the application server that no more work will be done on this connection, and to dissociate the Transaction branch from the connection.



**Figure 3-128** Ending a Global Transaction

Following are the valid XARETVALS that can be returned, when replying to an End request:

- XA\_OK                    Normal Execution.
- XAER\_RMERR            An error occurred associating the transaction branch to the XA protected connection.
- XAER\_RMFAIL           An error occurred that makes the XA protected connection unavailable.
- XAER\_PROTO            A protocol error has occurred.
- XAER\_NOTA             The XID is not known by the DBMS.
- XAER\_INVALID          Invalid XAFLAGS were specified.
- XA\_RDONLY             The DBMS is read-only and does not need to participate in the two-commit protocol. A one-phase optimization may be used instead.

**XA\_NODISSOCIATE** Protected resources associated with this transaction branch cannot be suspended, therefore the transaction branch will not be dissociated from the connection.

**XA\_NOMIGRATE** Protected resources associated with this transaction branch have been suspended but cannot be migrated to another connection. Transaction branch has been dissociated from the connection.

The returning of the XA\_RB\* will result in the transaction branch being marked as rollback-only, and the connection dissociated from the transaction branch.

**XA\_RBROLLBACK** The DBMS has marked the transaction branch as rollback-only for an unspecified reason.

**XA\_RBCOMMFAIL** A communication failure has occurred.

**XA\_RBDEADLOCK** The DBMS detected a deadlock.

**XA\_RBINTEGRITY** The DBMS detected a violation of the integrity of its resources.

**XA\_RBTIMEOUT** The work associated with this transaction branch took too long.

The relationship between the XA protected connection and XAFLAGS is as follows:

**TMSUCCESS** Indicates that the XA protected connection should be dissociated from the transaction branch that is currently associated with it. All cursors open at the application server and the application requester must be closed, except those marked with HOLD. And all protected RDB resources associated with the transaction branch (for example, temp tables) destroyed. Once the operation is complete, the XA connection is free to be associated with another transaction branch. Once all resources are destroyed or closed at both ends of the connection, the XID should be dissociated from the connection. The connection is now free to be associated with a different transaction branch. If the application requester had sent *RLSCONV(REUSE)* then the application server can reply back with *RLSCONV(REUSE)*, so that the connection can be pooled to a different application.

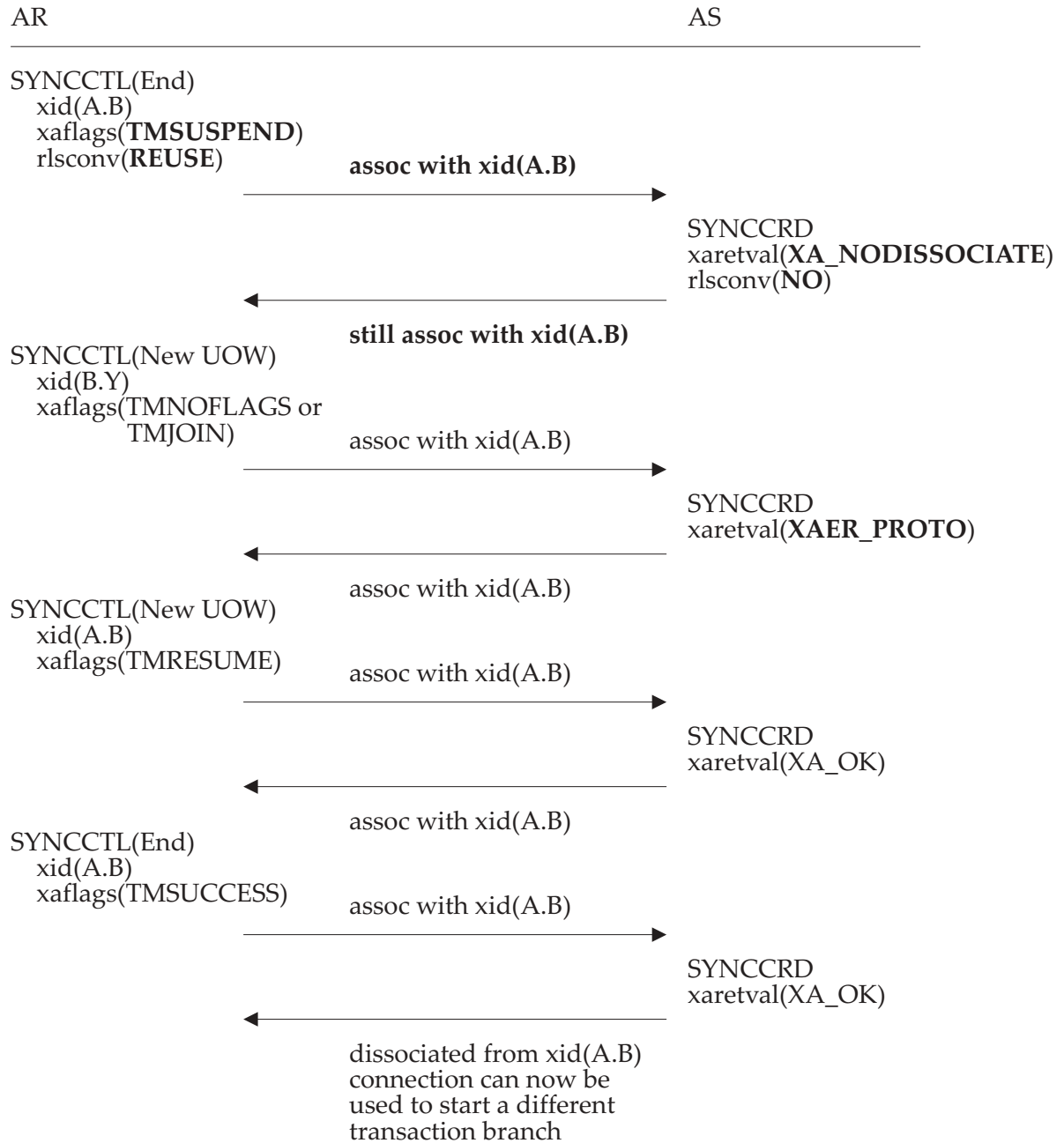
**Note:** It is OK to perform *SYNCCTL(End)* with *TMSUCCESS* on a branch which has already been sent *SYNCCTL(End)* with *TMSUSPEND* on the same connection. And the *SYNCCTL(End)* must be performed on the same connection where the transaction is being performed.

**TMSUSPEND** Indicates that the application is suspending the transaction branch and will resume the transaction branch later on this connection. The application server should suspend all open cursors associated with the transaction as well as all protected RDB resources associated with the transaction branch. (for example, temp tables). Once done, the application server should return *XA\_OK* and dissociate the XID from the connection. The application requester on receiving the reply should do likewise. The connection can now be associated with a different transaction. If application requester sent *rlsconv(resue)*, than the application server can reply with *rlsconv(reuse)*.

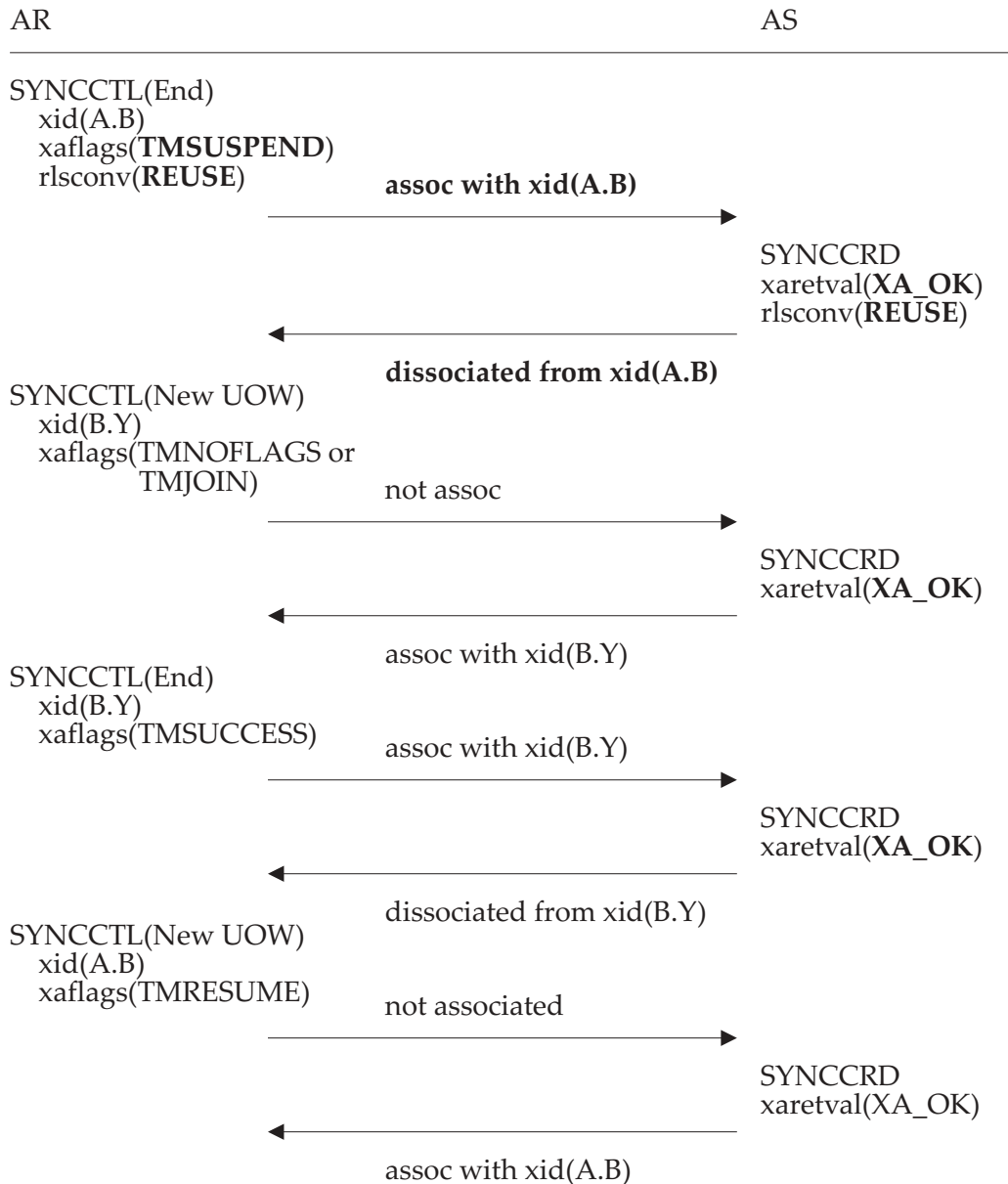
If the application server cannot suspend resources than it must reply back with *XARETVAL(XA\_NODISSOCIATE)*, and the connection should be kept associated with the transaction branch. If the application requester had sent *rlsconv(reuse)*, the application server must reply back with *rlsconv(no)*. On receiving the reply, the application requester should not

start a new transaction on the connection, or pool the connection to another application. The only operations allowed are RESUME, END with TMSUCCESS, or END with TMFAIL.

The figures below show examples of when the application server can and cannot suspend resources.



**Figure 3-129** Application Server Cannot Suspend Resources



**Figure 3-130** Application Server Can Suspend Resources

**TMSUSPEND with TMMIGRATE**

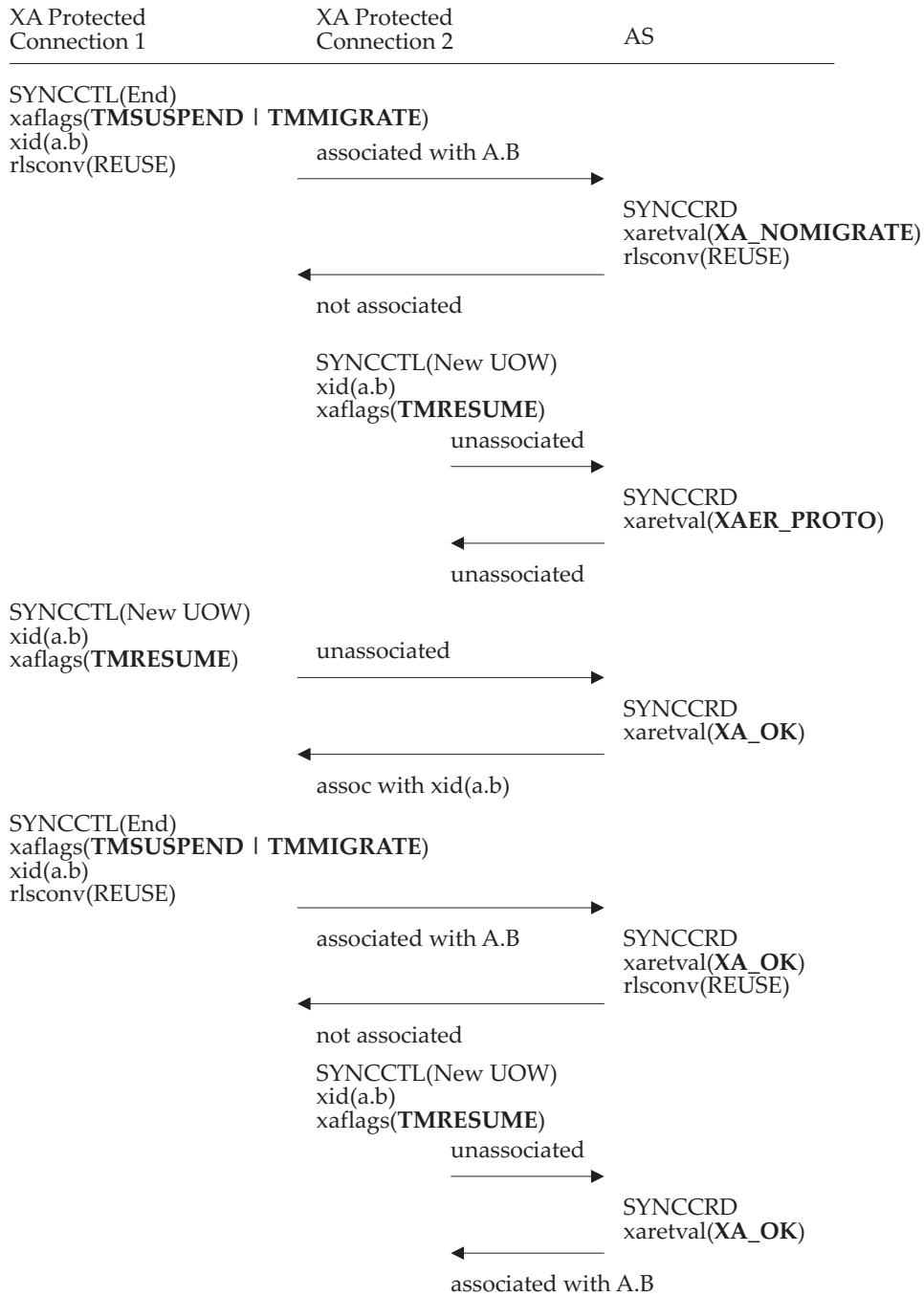
The application can only specify these flags together, which indicates that the application wants to suspend a transaction branch and resume it over a different connection. In order to do that, both the application requester and application server must first suspend all the open cursors and all protected RDB resources associated with the transaction branch (for example, temp tables), and then be able to migrate them to the connection that will be resuming the branch. Note the resume could be performed on the same connection. Both the application requester and application server must first be able to support TMSUSPEND. If the application server is unable to support TMSUSPEND, it should return



XA\_NODISSOCIATE indicating that it can do neither, and reply to *rlsconv(reuse)* with a NO. It should keep the connection associated with the suspended transaction branch.

If the application server can support TMSUSPEND but is unable to MIGRATE, then it should save all the open cursors and all protected RDB resources associated with the transaction branch (for example, temp tables), dissociate the connection from the XID, and reply with XA\_NOMIGRATE. XA\_NOMIGRATE indicates that application server was able to suspend the resources but can only resume the branch from this connection. If the application requester sent out *rlsconv(reuse)*, the application server can return *rlsconv(reuse)*.

If the application server can support both, then it should suspend the resources, dissociate the XID from the connection, and return XA\_OK. These resources should be resumed on the first connection that attempts to resume the branch. Note that it could be from the same connection. If the application requester sent *rlsconv(reuse)*, the application server can return *rlsconv(reuse)*.



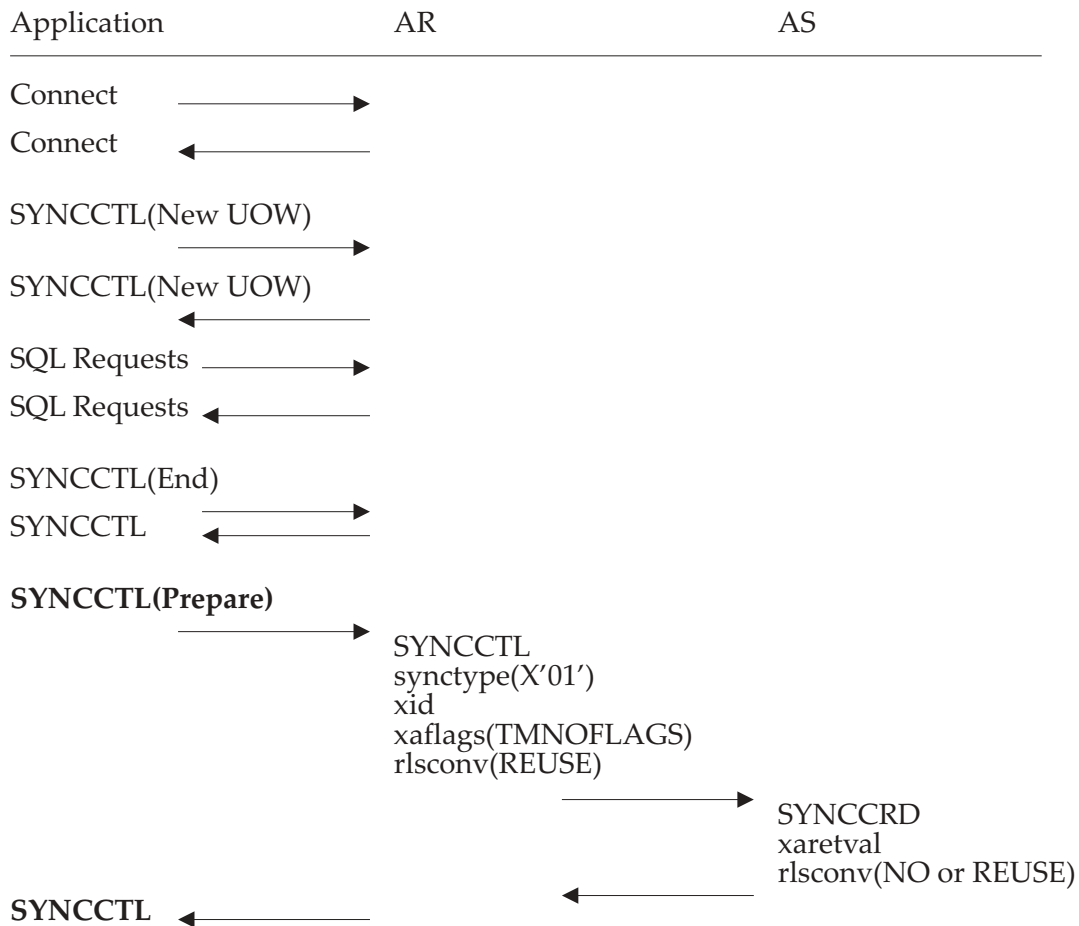
**Figure 3-131 Suspend and Migrate**

**TMFAIL**

Indicates that the TM has elected to fail this transaction branch, in which case the transaction is marked as rollback-only from this point. The XA protected connection is disassociated from the transaction branch and all open cursors including those marked with HOLD should be closed and all protected RDB resources associated with the transaction branch (for example, temp tables) destroyed.

**Preparing a Transaction (SYNCCTL - Prepare)**

Once the application has “successfully” ended all connections working on the same transaction branch, it is ready to commit the branch. The application starts the first phase of the commit process. The first phase involves sending the SYNCCTL(Prepare) to all resources involved in the transaction branch to prepare the branch for commit.



**Figure 3-132** Preparing a Global Transaction

Following are the valid XARETVALS that can be returned, when replying to a Prepare request:

- XA\_OK                      Normal Execution.
- XAER\_RMERR              An error occurred associating the transaction branch to the XA protected connection.
- XAER\_RMFAIL              An error occurred that makes the XA protected connection unavailable.
- XAER\_PROTO                A protocol error has occurred.
- XAER\_NOTA                 The XID is not known by the DBMS.
- XAER\_INVAL                Invalid XAFLAGS were specified.
- XA\_RDONLY                 The DBMS is read-only and does not need to participate in the two-commit protocol. A one-phase optimization may be used instead.

The returning of the XA\_RB\* will result in the transaction branch being rolled back, and all held resources released.

XA\_RBROLLBACK The DBMS has marked the transaction branch as rollback-only for an unspecified reason.

XA\_RBCOMMFAIL A communication failure has occurred.

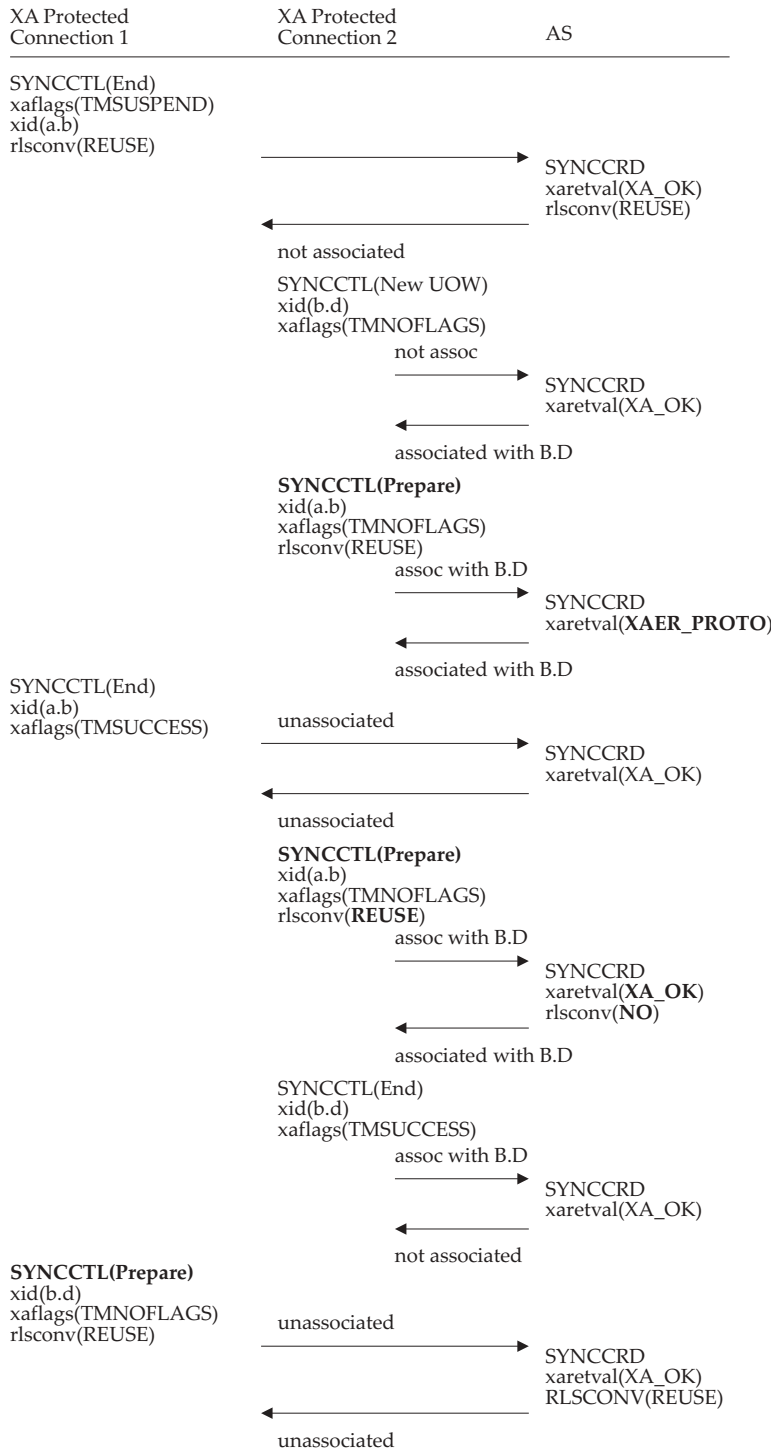
XA\_RBDEADLOCK The DBMS detected a deadlock.

XA\_RBINTEGRITY The DBMS detected a violation of the integrity of its resources.

XA\_RBTIMEOUT The work associated with this transaction branch took too long.

If the application requester sent out *rlsconv(reuse)* the application server must return *rlsconv(no)*, if the connection is currently associated with a branch. Otherwise, it can return *rlsconv(reuse)*. The application can prepare a transaction branch on any connection to that application server, even though that connection may be associated with a different transaction branch.

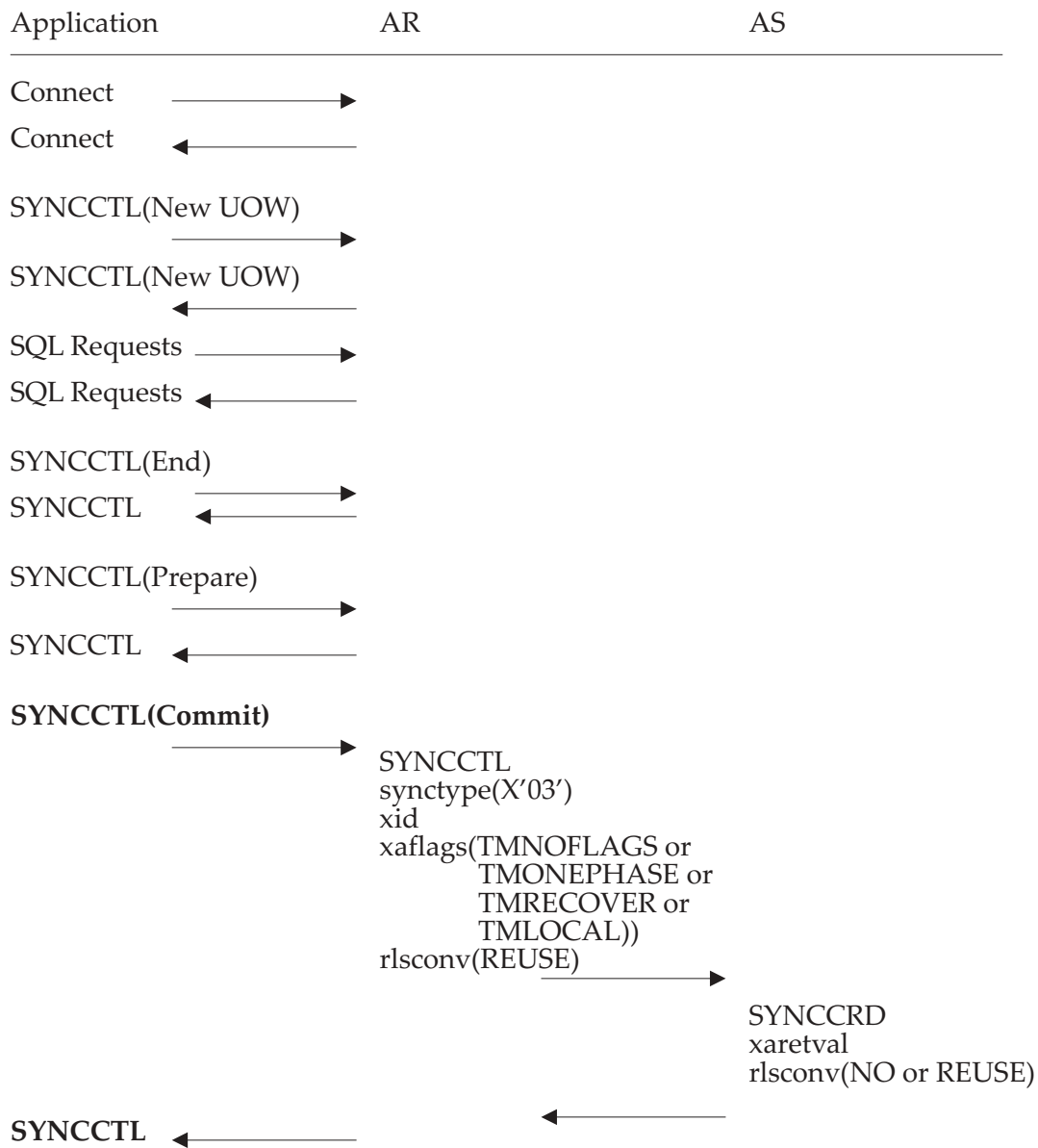
**Note:** All branches have to be Ended with TMSUCCESS before they can be Prepared. The Prepare does not have to be performed on the same connection as the SYNCCTL(END).



**Figure 3-133** Example of a Prepare from any Connection

**Committing a Transaction (SYNCCTL- Commit)**

The application initiates the second phase of the two-phase protocol, once all resources involved in the Global Transaction have successfully prepared the transaction. The application sends SYNCCTL(Commit) to commit the Global Transaction. Or issues the SYNCCTL to commit a Local Transaction. If the application requester sent out *rlsconv(reuse)*, the application server must return *rlsconv(no)*, if the connection is currently associated with a branch. Otherwise, it can return *rlsconv(reuse)*.



**Figure 3-134** Committing a Transaction

Following are the valid XARETVALS that can be returned, when replying to a Commit request:

XA\_OK                      Normal Execution.

XAER_RMERR	An error occurred associating the transaction branch to the XA protected connection.
XAER_RMFAIL	An error occurred that makes the XA protected connection unavailable.
XAER_PROTO	A protocol error has occurred.
XAER_NOTA	The XID is not known by the application server.
XAER_INVALID	Invalid XAFLAGS were specified.
XA_TWOPHASE	The application server requires two-phase commit. The application must restart the commit procedure using two-phase protocols.
XA_RETRY_COMMFAIL	Due to communication failures with subordinates, the application server or database server was not able to commit the transaction. Re-issue the Commit at a later time. All held resources on behalf of the transaction branch remain in a prepared state until commitment is possible.
XA_RETRY	Due to general failure—for example, out of storage—the application server or database server was unable to commit the transaction branch. Re-issue the commit at a later time. All held resources on behalf of the transaction branch remain in a prepared state until commitment is possible.
XA_HEURHAZ	Due to failure, it is not known whether all subordinates have performed the same operation (commit or rollback). One or more may be in danger of making an inconsistent heuristic decision.
XA_HEURCOM	Due to a heuristic decision, the specified transaction branch was committed.
XA_HEURRB	Due to a heuristic decision, the specified transaction branch was rolled back.
XA_HEURMIX	Due to a heuristic decision, the specified transaction branch was partially committed and partially rolled back.
The returning of the XA_RB* will result in the transaction branch being rolled back, and all held resources released. These return values can only be sent if the XAFLAG ('TMONEPHASE') was specified.	
XA_RBROLLBACK	The DBMS has marked the transaction branch as rollback-only for an unspecified reason.
XA_RBCOMMFAIL	A communication failure has occurred.
XA_RBDEADLOCK	The DBMS detected a deadlock.
XA_RBINTEGRITY	The DBMS detected a violation of the integrity of its resources.
XA_RBTIMEOUT	The work associated with this transaction branch took too long.
The relationship between the XA protected connection and XAFLAGS is as follows:	
<b>Note:</b>	SYNCCCTL(Committed) can be performed from any XA protected connection.
TMNOFLAGS	Indicates that all work done under this transaction branch should be committed. All cursors should be closed on both sides and all protected RDB resources associated with the transaction branch (for example, temp tables) destroyed. The XA connection should be dissociated, so that a different transaction branch may be associated with it.

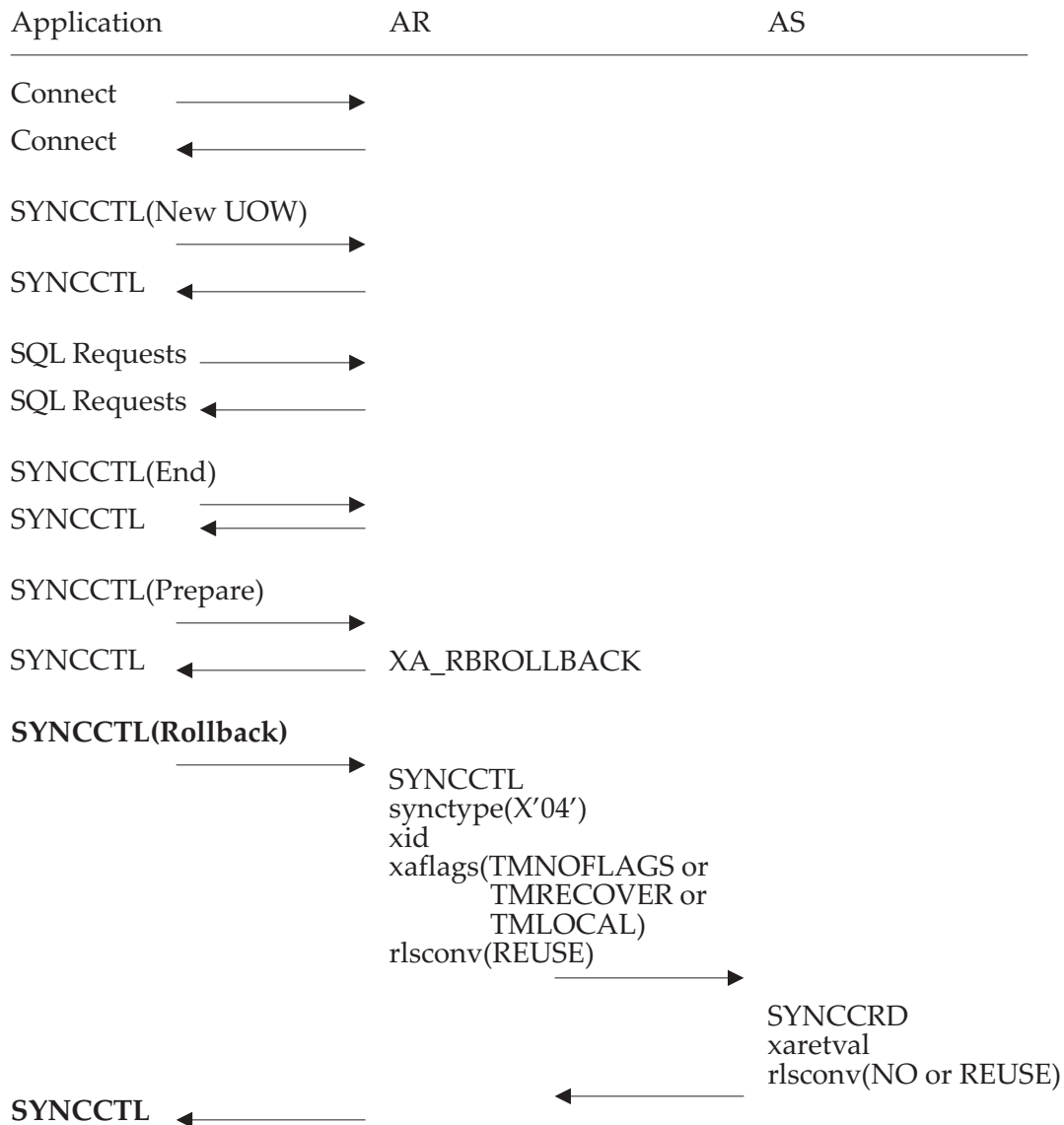
- TMONEPHASE**      Indicates that the application is using a one-phase commit optimization; no SYNCCTL(Prepare) was sent out for this transaction branch. All cursors should be closed on both sides and all protected RDB resources associated with the transaction branch (for example, temp tables) destroyed.
- TMRECOVER**      Indicates that the application is attempting to commit a transaction branch. See the Recovery section for more details.

The application can commit a transaction branch on any connection to that application server, even though that connection may be associated with a different transaction branch.



**Rolling Back a Transaction (SYNCCTL - Rollback)**

If the first phase of the two-phase protocol fails or a failure has occurred, the application issues a SYNCCTL(Rollback) to roll back the Global Transaction. It also uses the same SYNCTYPE to roll back a Local Transaction. If the application requester sent out *rlsconv(reuse)*, the application server must return *rlsconv(no)*, if the connection is currently associated with a branch. Otherwise, it can return *rlsconv(reuse)*.



**Figure 3-135** Rolling Back a Transaction

Following are the valid XARETVALS that can be returned, when replying to a Rollback request:

- XA\_OK                      Normal Execution.
- XAER\_RMERR              An error occurred associating the transaction branch to the XA protected connection.

XAER_RMFAIL	An error occurred that makes the XA protected connection unavailable.
XAER_PROTO	A protocol error has occurred.
XAER_NOTA	The XID is not known by the application server.
XAER_INVAL	Invalid XAFLAGS were specified.
XA_RETRY_COMMFAIL	Due to communication failures with subordinates, the application server or database server was not able to commit the transaction. Re-issue the Rollback at a later time. All held resources on behalf of the transaction branch remain in a prepared state until rollback is possible.
XA_HEURHAZ	Due to failure, it is not known whether all subordinates have performed the same operation (commit or rollback). One or more may be in danger of making an inconsistent heuristic decision.
XA_HEURCOM	Due to a heuristic decision, the specified transaction branch was committed.
XA_HEURRB	Due to a heuristic decision, the specified transaction branch was rolled back.
XA_HEURMIX	Due to a heuristic decision, the specified transaction branch was partially committed and partially rolled back.

The returning of the XA\_RB\* will result in the transaction branch being rolled back, and all held resources released. These return values are typically sent when the transaction branch has already been marked as rollback.

XA_RBROLLBACK	The DBMS has marked the transaction branch as rollback-only for an unspecified reason.
XA_RBCOMMFAIL	A communication failure has occurred.
XA_RBDEADLOCK	The DBMS detected a deadlock.
XA_RBINTEGRITY	The DBMS detected a violation of the integrity of its resources.
XA_RBTIMEOUT	The work associated with this transaction branch took too long.

The relationship between the XA protected connection and XAFLAGS is as follows:

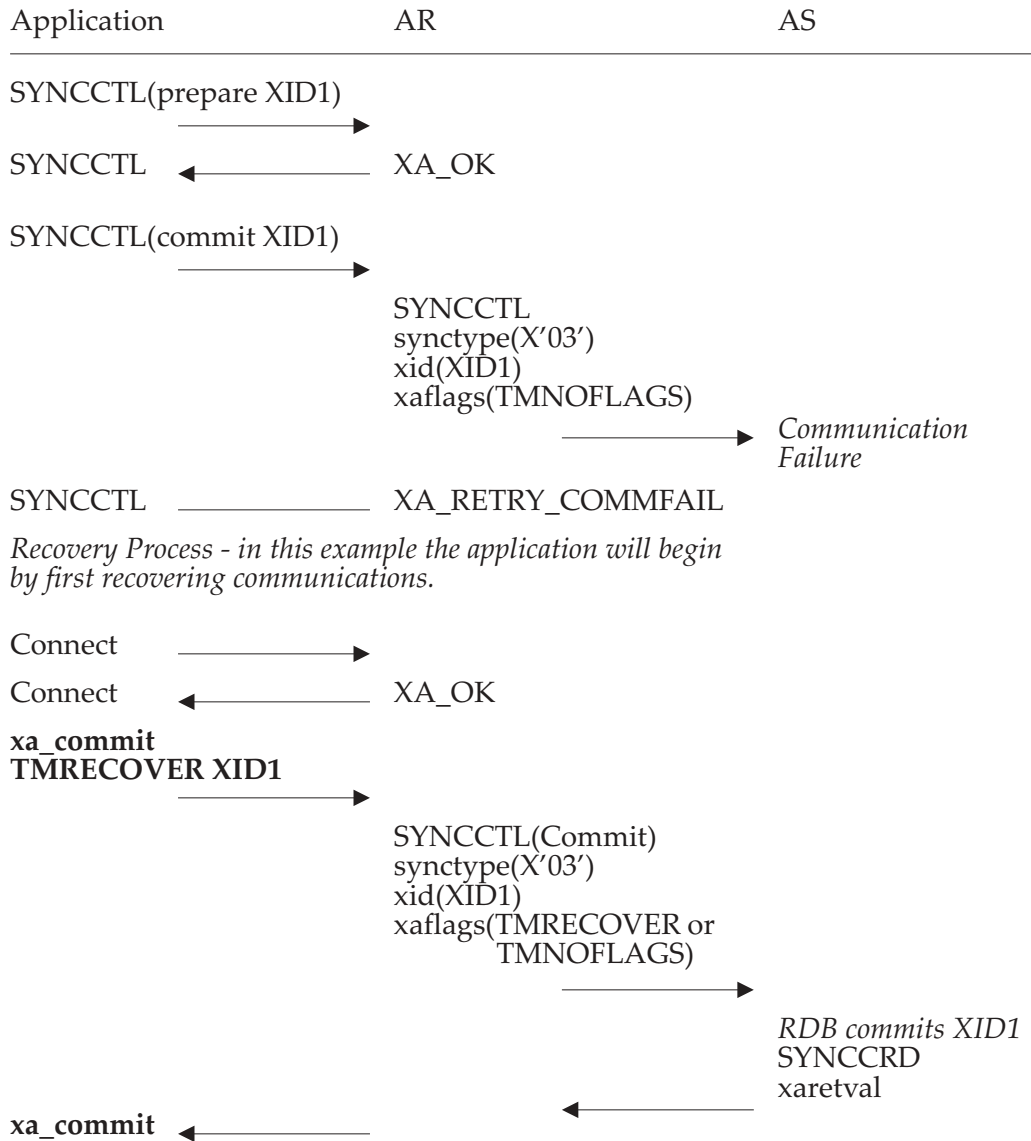
**Note:** SYNCCTL(Rollback) can be performed from any XA protected connection.

TMNOFLAGS	Indicates that all work done under this transaction branch should be rolled back. All cursors should be closed on both sides and all protected RDB resources associated with the transaction branch (for example, temp tables) destroyed. The XA connection should be dissociated so that a different transaction branch may be associated with the XA connection.
TMRECOVER	Indicates the application is attempting to roll back a transaction branch. See the Recovery section for more details.

The application can roll back a transaction branch on any connection to that application server, even though that connection may be associated with a different transaction branch.

**Recovery**

The application is responsible for all recovery operations when failures occur, when the application is performing the second phase coordination to the DBMSs involved in the same Global Transaction. These failures could be the result of a communication failure between the application requester and application server. In any case the application will drive recovery using the TMRECOVER flags on SYNCCTL(X'03' and X'04'). The flows below outline their use.



**Figure 3-136** Recovery Process

**Note:** If the second phase was trying to perform a rollback, then the application would issue SYNCCTL(Rollback) (TMRECOVER, XID) after recovering communications with the application server.

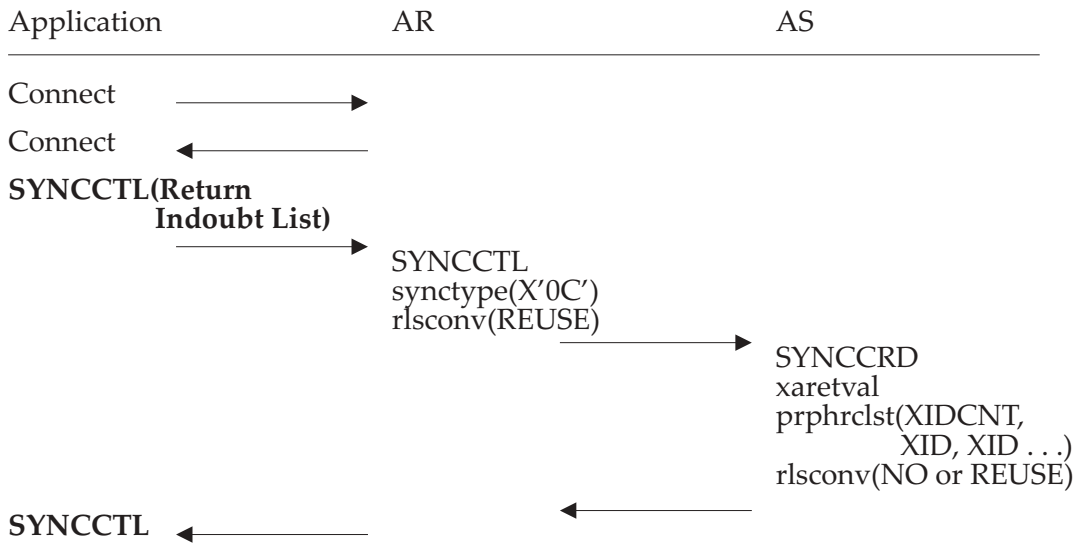
The TMRECOVER is an optional flag; the application may elect not to use it.

Following are the valid XARETVALS that can be returned, when replying to a Recover request:

XA_OK	Normal Execution.
XAER_RMERR	An error occurred associating the transaction branch to the XA protected connection.
XAER_RMFAIL	An error occurred that makes the XA protected connection unavailable.
XAER_PROTO	A protocol error has occurred.

**Prepared or Heuristically Completed Transactions**

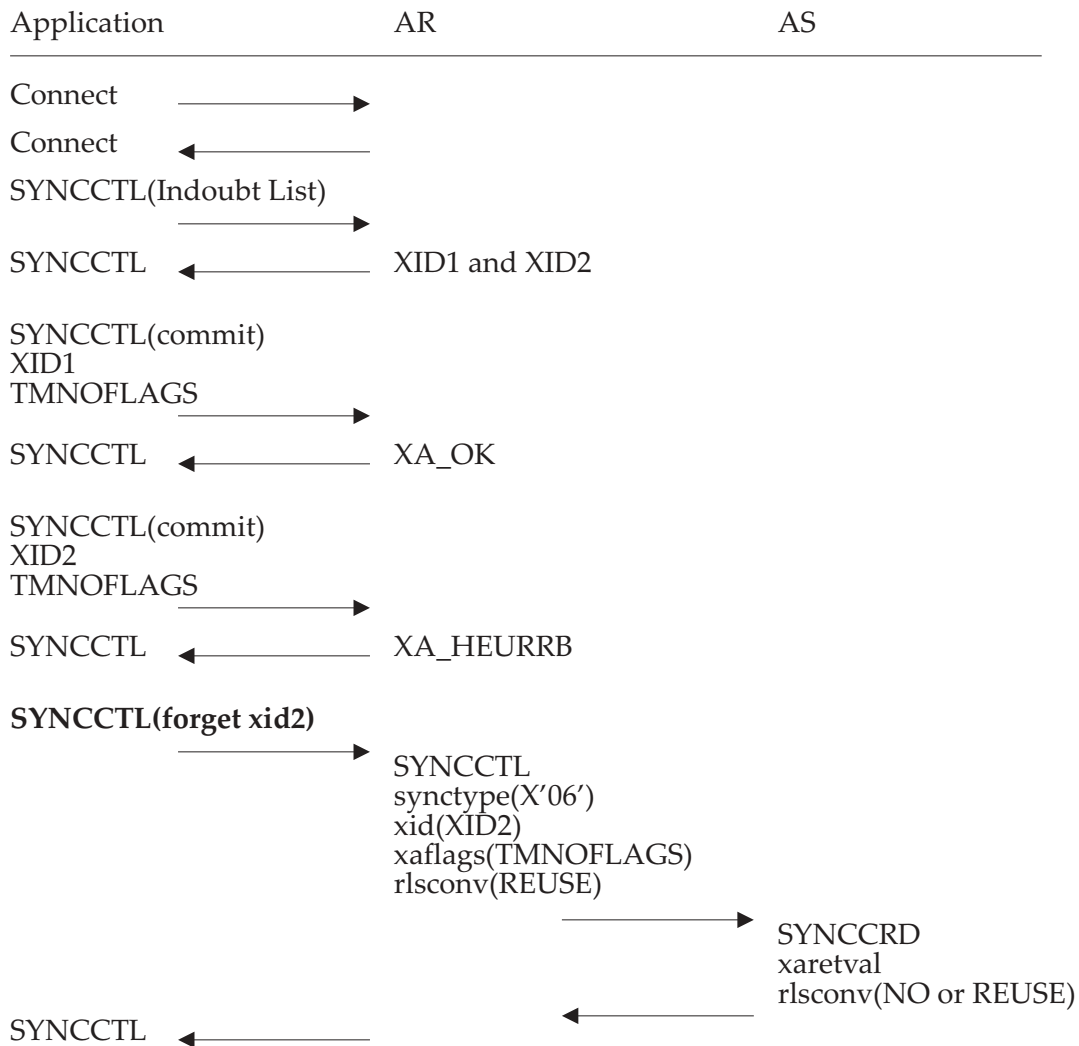
The application may want to know at a given time which transaction branches at the DBMS are currently prepared or heuristically completed. The application does this by sending SYNCCTL(X'0C') which obtains a list of prepared or heuristically completed transaction branches at the present time on the DBMS. Note two consecutive SYNCCTL(X'0C') requests should return the same list of transaction branches, unless the application has performed SYNCCTL(Commit)ted, SYNCCTL(Prepare)d, SYNCCTL(Forget), XARolledback, or the DBMS has heuristically completed some branches between the two SYNCCTL(X'0C') requests. The flow outlines the use of the SYNCCTL(X'0C') request in obtaining the list of prepared or heuristically completed transactions.



**Figure 3-137** Obtaining a List of XIDs from the Application Server

For prepared transaction branches, the application uses the normal method to commit or roll back the transaction. Heuristically completed transactions on the other hand have either been committed, rolled back, or a mixture of both by the DBMS. The DBMS informs the application with the XA\_HEUR values. Because the DBMS made a heuristic decision in committing or rolling back the transaction branch independent of the application, it must keep the knowledge of the transaction branch. The transaction branch cannot be reused or associated until the application authorizes the DBMS to forget about the branch. The application does this by using the SYNCCTL(Forget). If the application requester sent out rlsconv(reuse), the application server must return rlsconv(no), if the connection is currently associated with a branch. Otherwise, it can return rlsconv(reuse). The application can issue the recover request from any connection to that application server, even though that connection may be associated with a transaction branch.

The sample belows shows the application performing SYNCCTL(Return Indoubt List) and receiving two XIDs, which it attempts to commit. Because the second transaction branch was heuristically rolled back, the application server informs the application thorough the XA return value. The application then authorizes the DBMS to forget the transaction branch.



**Figure 3-138** Forgetting Heuristically Completed Transaction Branches

**Note:** The DBMS must always inform the application of the heuristic decision even if the request matches the application’s decision; that is, the application may have elected to perform SYNCCTL(Rollback) on XID2 after the SYNCCTL(Return Indoubt List), and the result of the SYNCCTL(Rollback) should result in an XA\_HEURRB being returned. SYNCCTL(Forget) should only be used if the application server indicated to the application requester that the transaction branch was heuristically completed.

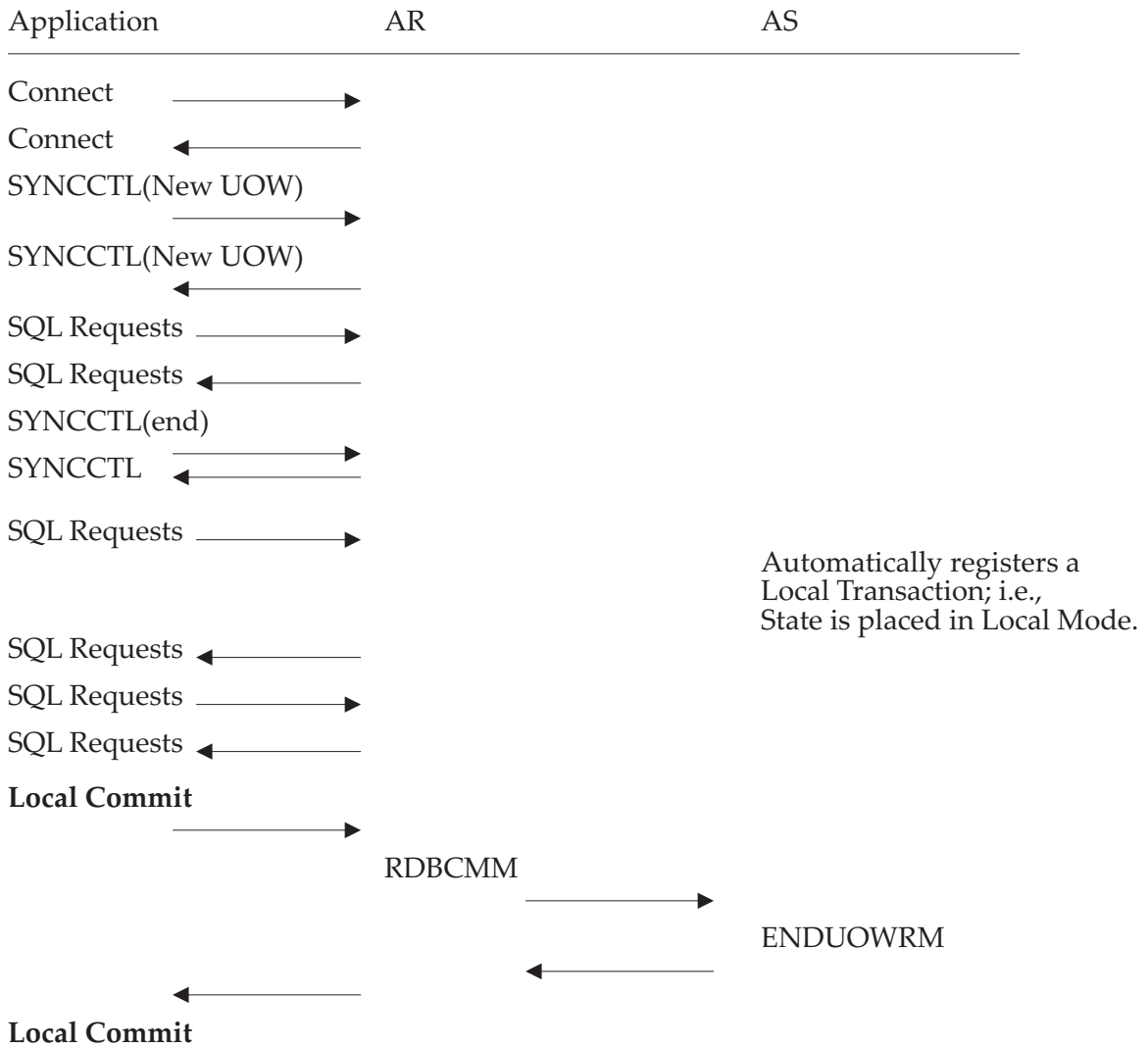
Following are the valid XARETVALS that can be returned, when replying to a Forget request:

- XA\_OK                      Normal Execution.
- XAER\_RMERR              An error occurred associating the transaction branch to the XA protected connection.
- XAER\_RMFAIL             An error occurred that makes the XA protected connection unavailable.
- XAER\_PROTO              A protocol error has occurred.
- XAER\_NOTA                The XID is not known by the application server.

XAER\_INVAL Invalid XAFLAGS were specified.

**Local Transactions**

Local Transactions provide an application with the mechanism to perform an unprotected unit of work to a DBMS over an XA protected connection. The application does not need to register the Local Transaction. The DRDA application server will automatically register the Local Transaction when the first SQL is issued. An application server that is a Local Transaction should not expect an end, prepare, recover, or forget for this transaction. If the application server receives any of these requests, it should reject them with a *prccnvcd* (X'11'). The application will use RDBCMM and RDBRLLBCK to perform a static commit or rollback. It is valid to use dynamic methods in committing or rolling back a Local Transaction.



**Figure 3-139** Example of a Local Transaction

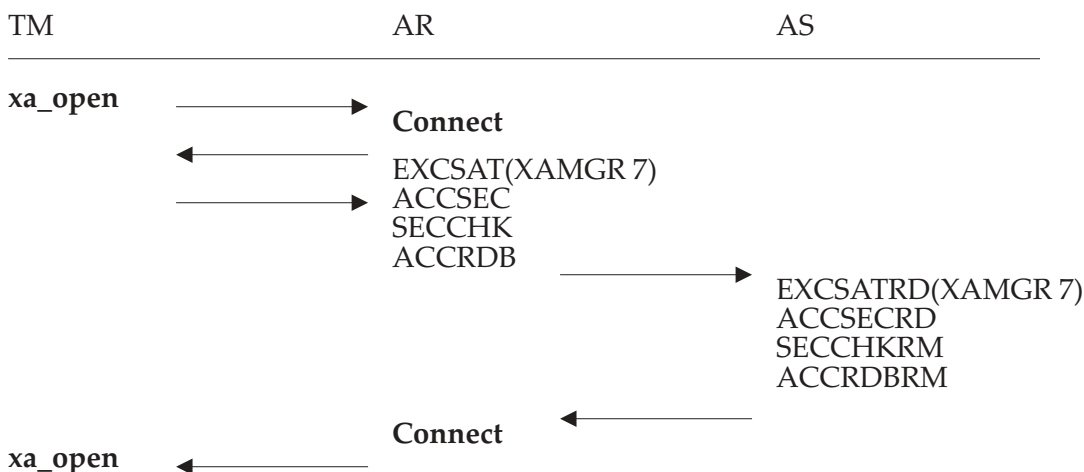
Once the commit or rollback has been performed either statically or dynamically, the connection is available to start a new Global Transaction or another Local Transaction. If the application requester sent *rlsconv*(reuse) at commit or rollback, the application server can reply with reuse.

**Implementing the XA Interface using DRDA TP Support**

The DRDA TP support is modeled after the XA protocol, and can be used by a Resource Manager to implement an XA interface. The following shows how each XA API would use the DRDA TP support. In our case a Resource Manager is the application that provides the TM with a set of XA APIs and communicates these XA operations to a DBMS. If the DBMS is using the DRDA protocol, the application must use the XAMGR to convey these XA requests to the application server or database server of the DBMS.

*xa\_open* This API is used to initialize a reply message, and could involve forming a connection. This sample shows a flow where the *xa\_open* API is requesting XAMGR level support.

**Note:** The application acting as a reply message may elect to connect later.

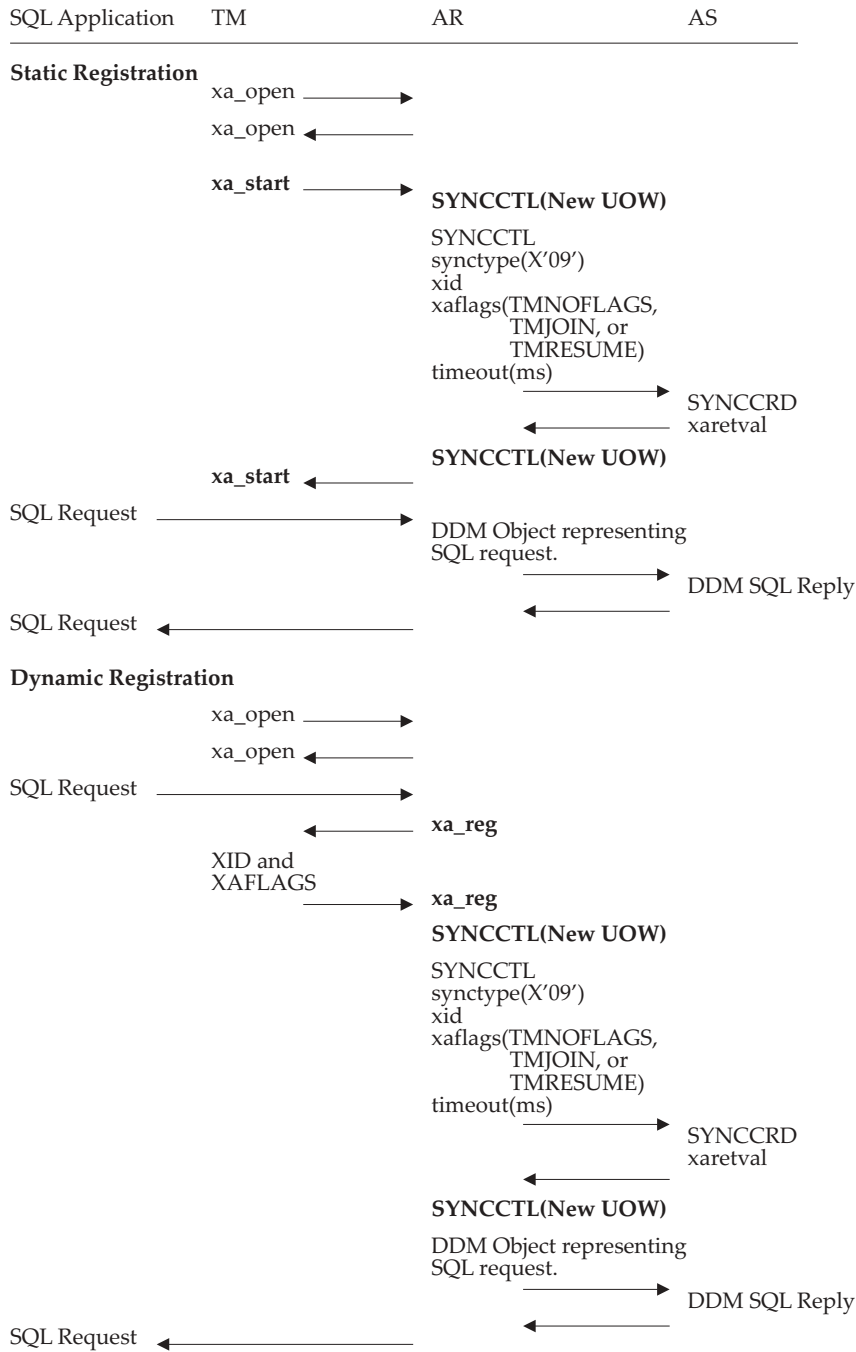


**Figure 3-140** *xa\_open* Requesting XAMGR Support

*xa\_start* and *ax\_reg*

These APIs are used by the TM to register a transaction with an RM. There are two modes of registration: *static* and *dynamic*. Static registration involves the TM calling the *xa\_start* API to inform the RM that the SQL application may do work on behalf of a transaction branch. Dynamic registration involves the RM calling the *ax\_reg* API to inform the TM that an SQL application is about to perform work. The difference between the two is that, with dynamic the transaction is only registered with the TM when work is actually performed on the RM. If no work is done on the RM, then the RM never registers with the TM and hence the RM does not have to be involved in two-phase commit coordination. While static registration does not guarantee that a transaction will occur because the RM does not inform the TM whether or not any work was done on the connection. Hence, the TM will still perform two-phase commit on the RM. Below is the flow outlining both modes and how they would use SYNCCTL(New UOW).

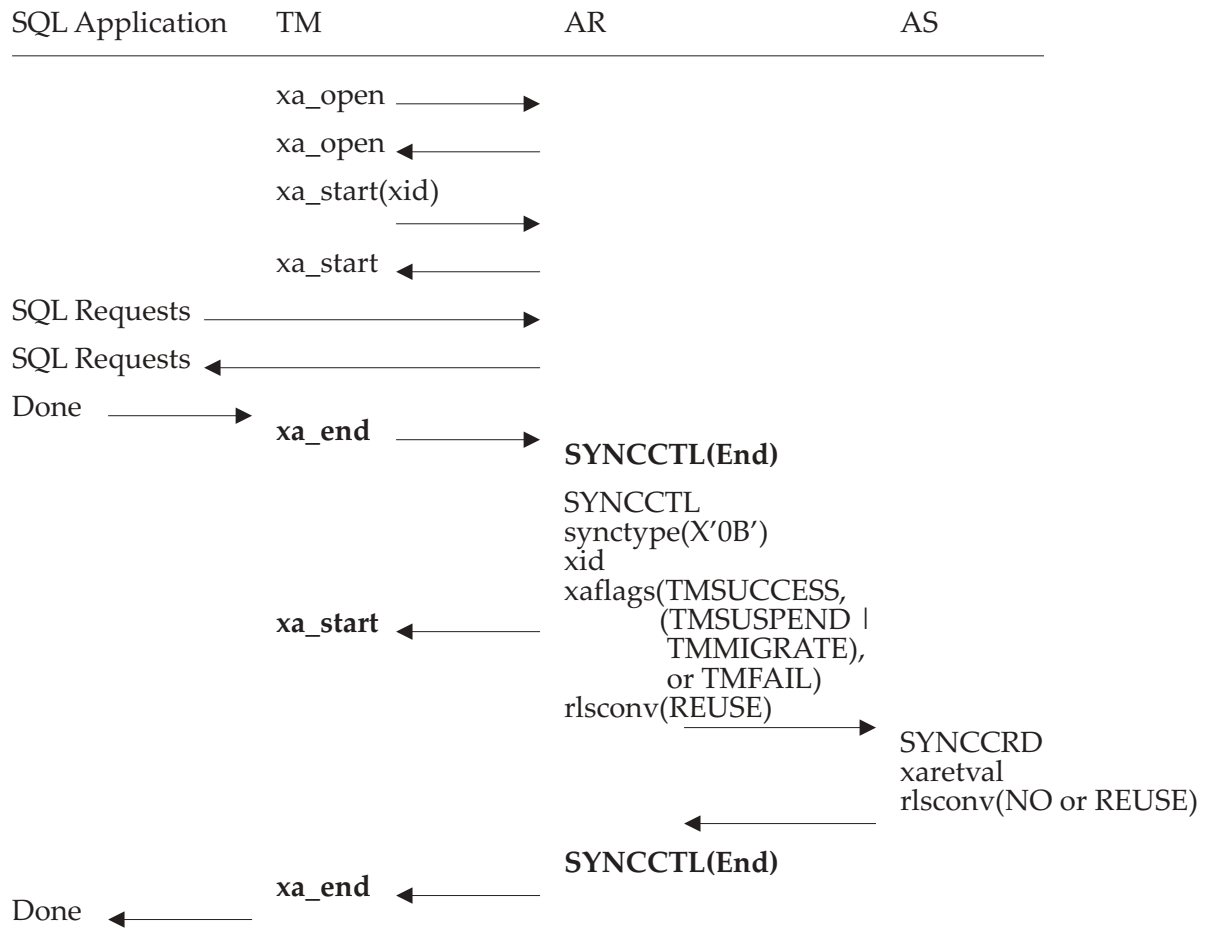




**Figure 3-141** Sample of Static and Dynamic Registration using SYNCCTL(New UOW)

*xa\_end*

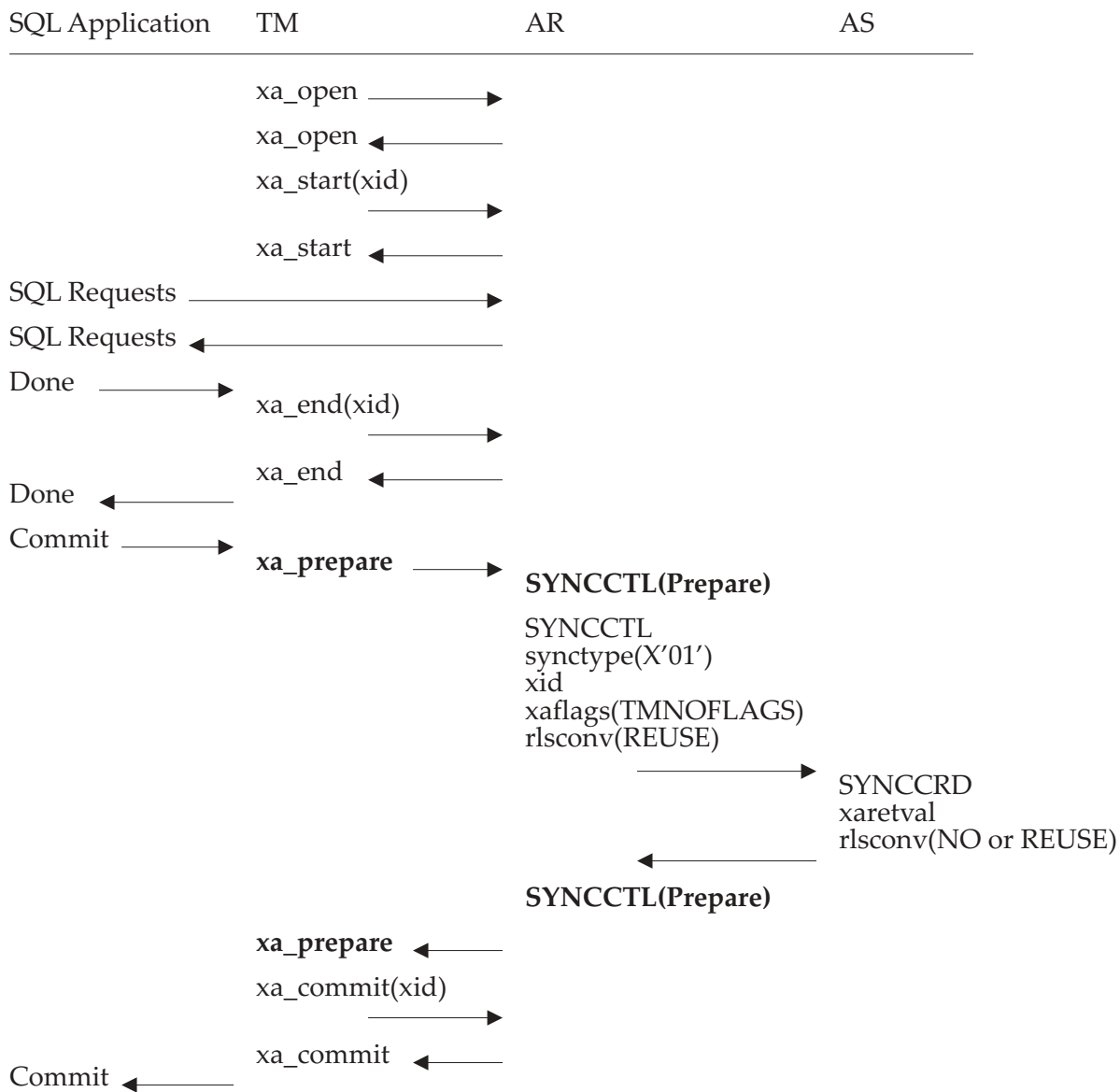
The TM calls the *xa\_end* API to end or suspend the transaction branch. This occurs when the SQL application has completed its work, either partially or in its entirety. The SQL application informs the TM, which initiates the call to *xa\_end*. The sample below shows how *xa\_end* would use SYNCCTL(End).



**Figure 3-142** Sample Flow of `xa_end` using `SYNCCTL(End)`

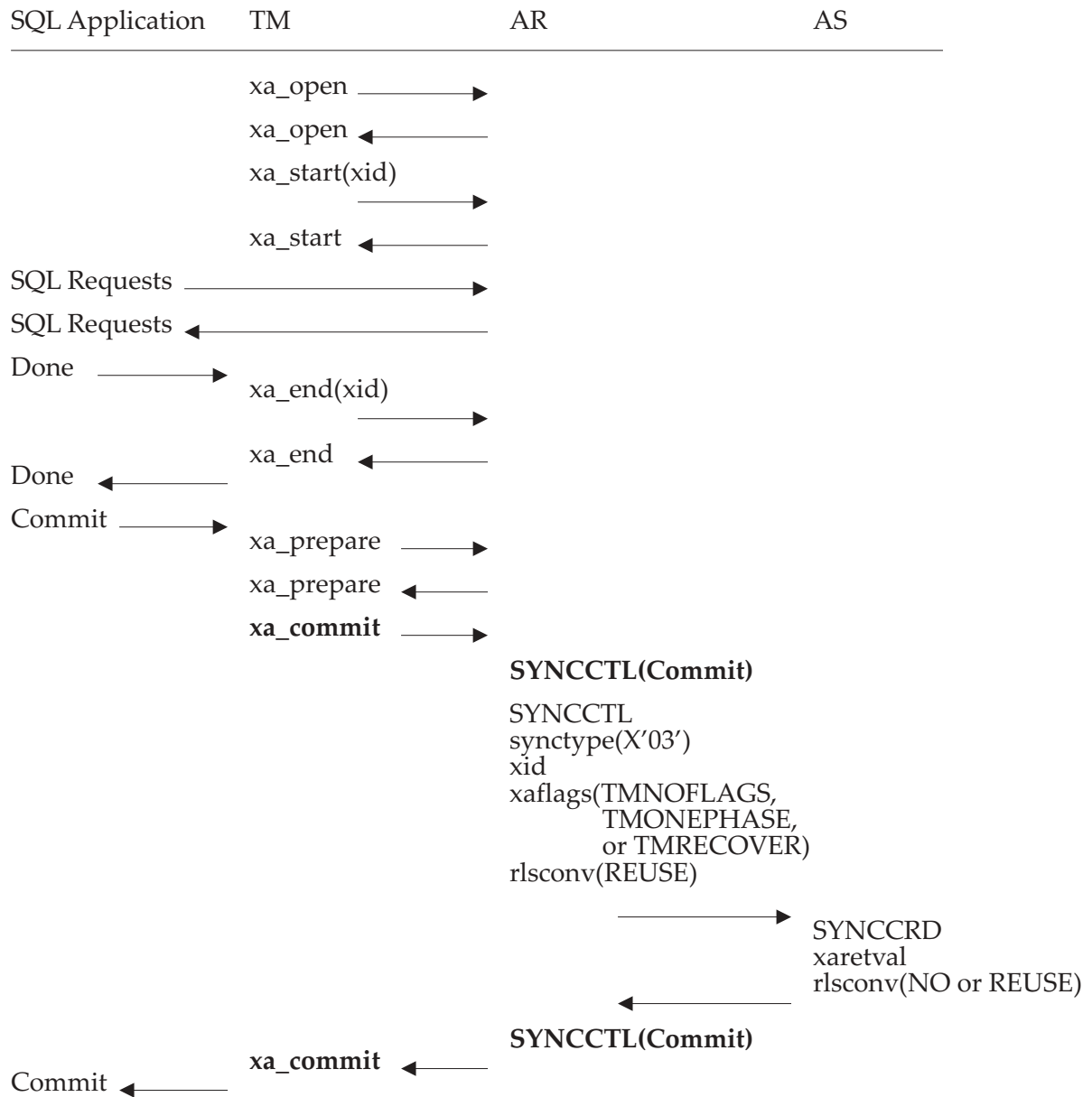
*xa\_prepare*

The TM calls the `xa_prepare` API to request the RM to prepare to commit the work done on behalf of a transaction branch. Below is a flow showing how `xa_prepare` would use `SYNCCTL(Prepare)`.



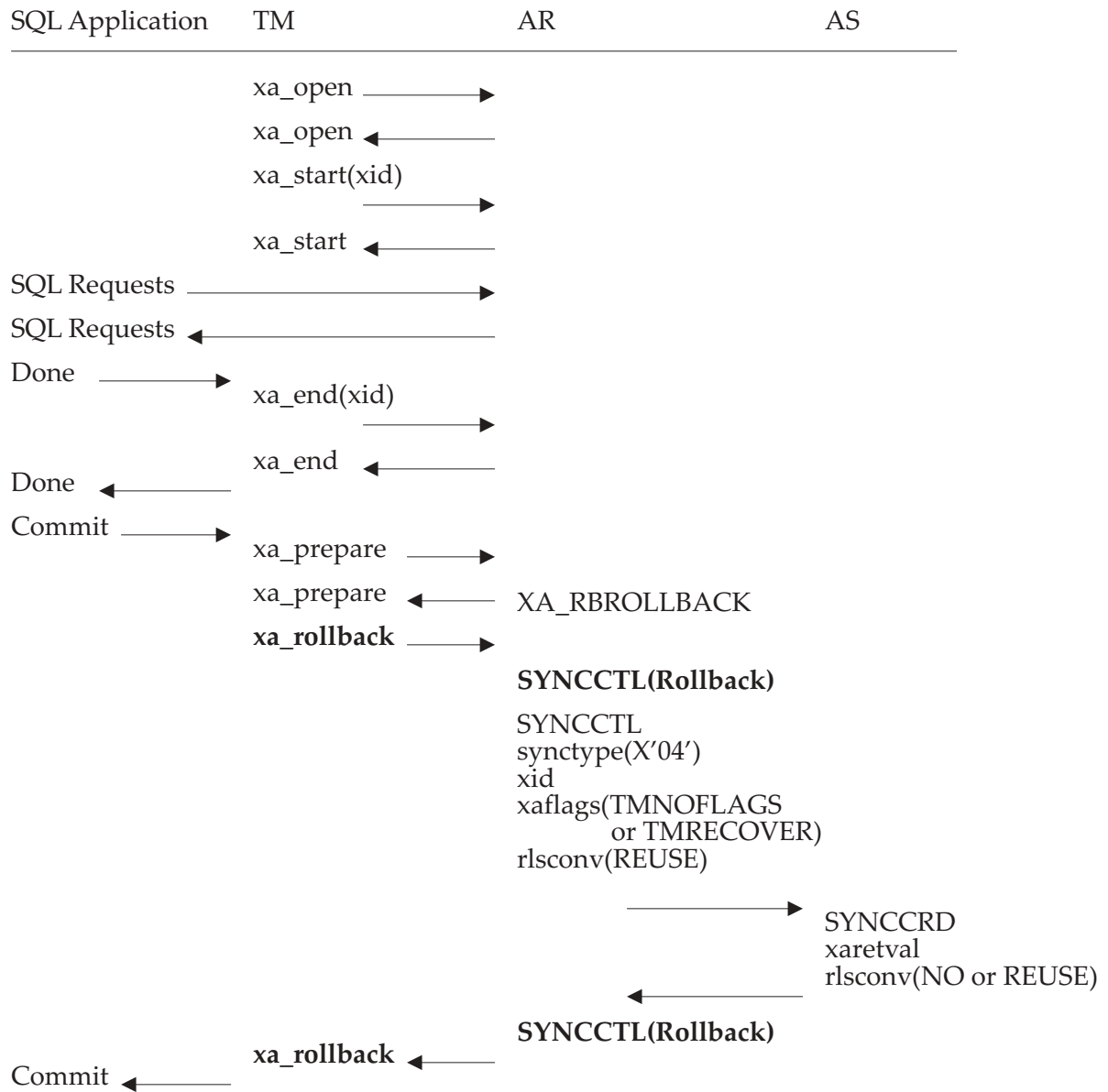
**Figure 3-143** Sample Flow of `xa_prepare` using `SYNCCTL(Prepare)`

**xa\_commit** The TM calls the `xa_commit` API to request the RM to commit the work done on behalf of a transaction branch. Below is a flow showing how `xa_commit` would use `SYNCCTL(Commit)`.



**Figure 3-144** Sample flow of `xa_commit` using `SYNCCTL(Commit)`

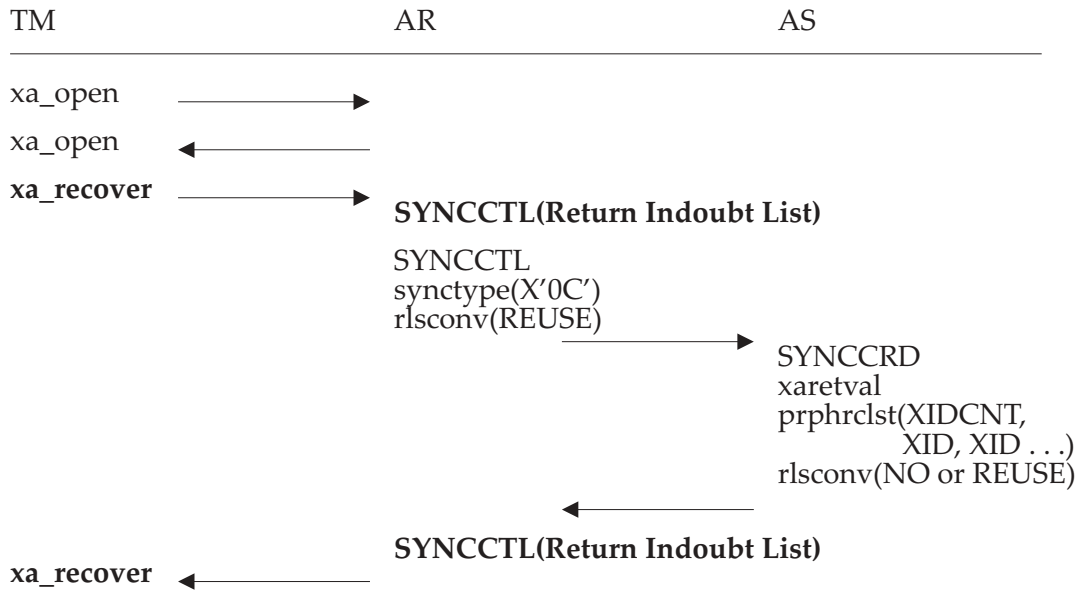
***xa\_rollback*** The TM calls the `xa_rollback` API to request the RM to roll back the work done on behalf of a transaction branch. Below is a flow showing how `xa_rollback` would use `SYNCCTL(Rollback)`.



**Figure 3-145** Sample Flow of xa\_rollback using SYNCCTL(Rollback)

*xa\_recover*

The TM may want to know at a given time which transaction branches at the RM are currently prepared or heuristically completed. The TM does this by calling the *xa\_recover* API which obtains a list of prepared or heuristically completed transaction branches at the present time on the RM. Below is a flow showing how *xa\_recover* would use SYNCCTL(Return Indoubt List).

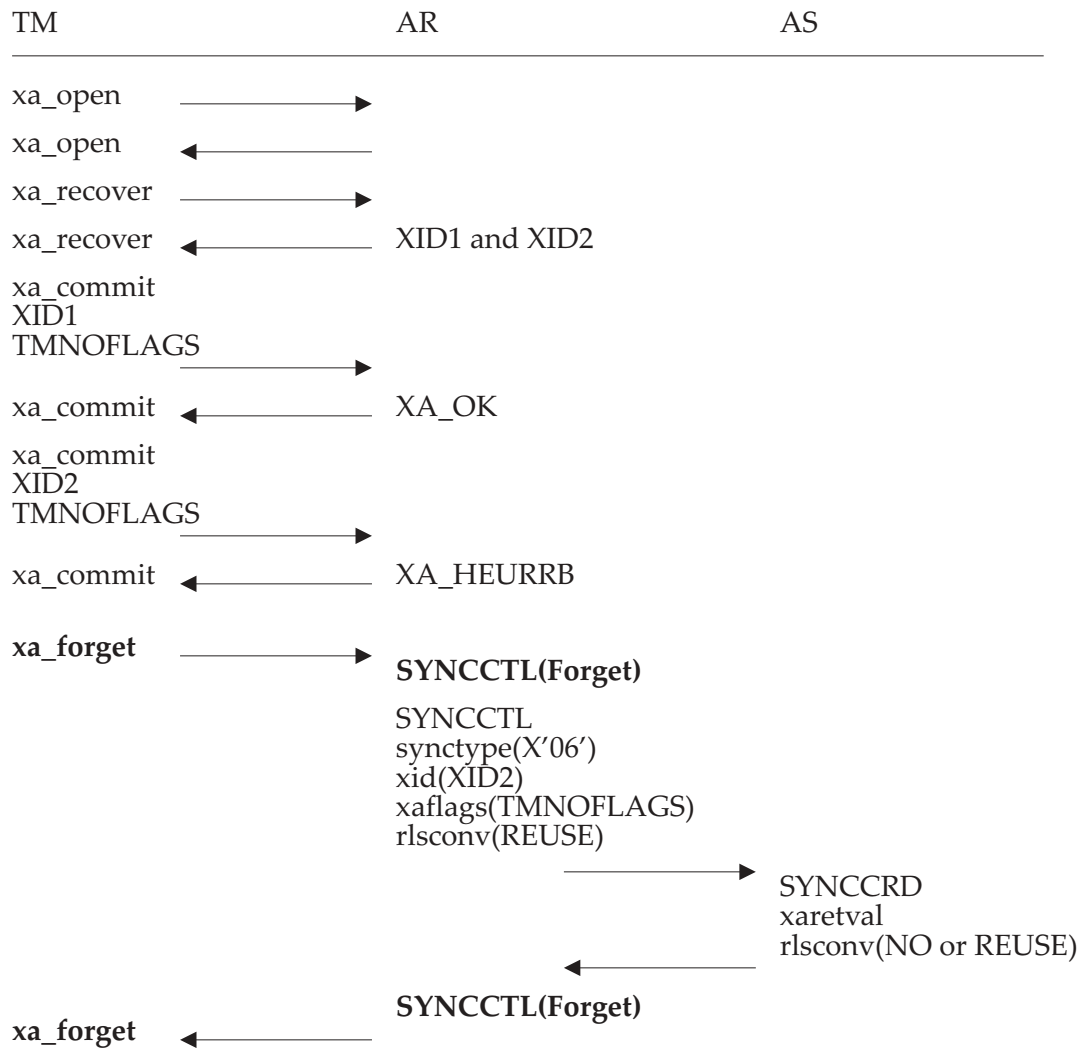


**Figure 3-146** Sample Flow of xa\_recover using SYNCCTL(Return Indoubt List)

*xa\_forget*

For prepared transaction branches, the TM use the normal method to commit or roll back the transaction. Heuristically completed transactions on the other hand have been either committed, rolled back, or a mixture of both by the RM. The RM informs the TM with the XA\_HEUR values. Because the RM made a heuristic decision in committing or rolling back the transaction branch independent of the TM, it must keep the knowledge of the transaction branch. The transaction branch cannot be reused or associated until the TM authorizes the RM to forget about the branch.

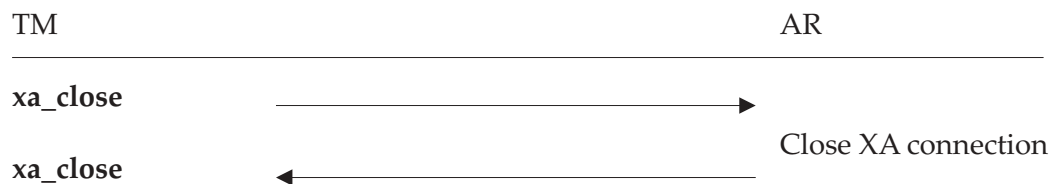
The TM does this by using the *xa\_forget* API. Below is a flow showing how *xa\_forget* would use SYNCCTL(Forget).



**Figure 3-147** Sample Flow of xa\_forget using SYNCCTL(Forget)

*xa\_close* is used by the XA TM to close a currently opened RM. Once closed, that RM must be re-opened if it is to participate in any Global Transactions.

**Note:** The *xa\_close* API has no bearing on the DRDA flows between the source and target servers.

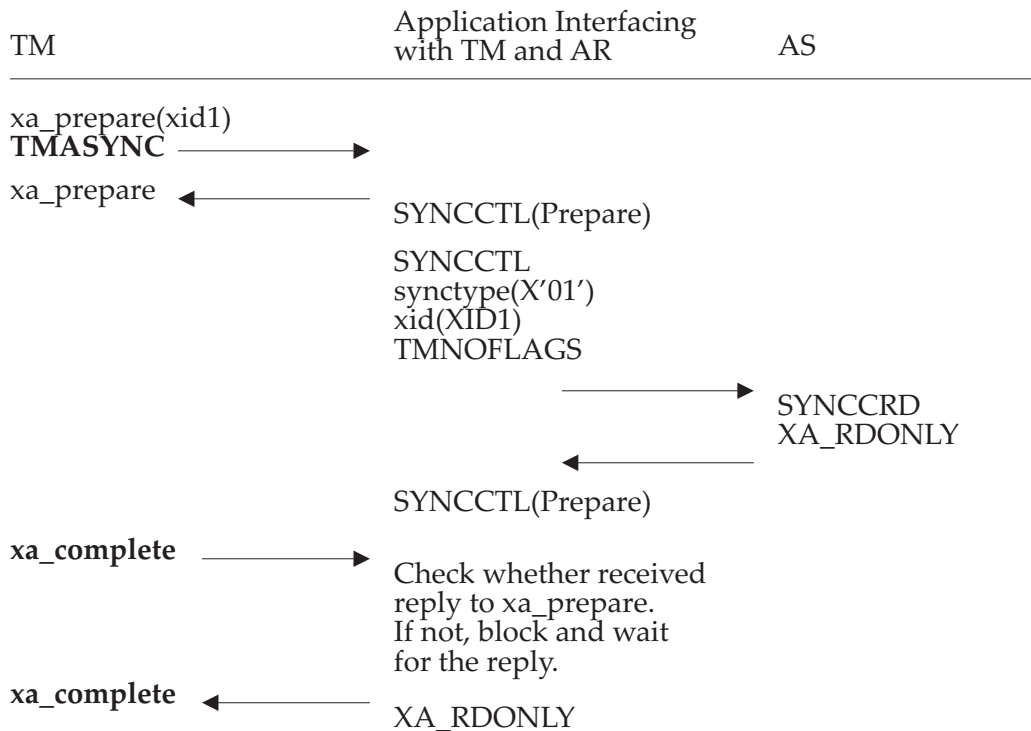


**Figure 3-148** xa\_close

**Asynchronous Operations**

The XA protocol does allow its operations to be completed asynchronously using the flag TMASYNC and the *xa\_complete* API. This flag and the *xa\_complete* API have no bearing on the DRDA flows between the source and target servers.

For thoroughness, the flow below outlines asynchronous operations. See *DTP: The XA+ Specification* for which APIs allow the TMASYNC flag.



**Figure 3-149** Asynchronous XA Operation

**SEE ALSO**

- Semantic** *RDB* (on page 718)
- SYNCCRD* (on page 913)
- SYNCCTL* (on page 915)
- XAFLAGS* (on page 1061)
- XAMGR* (on page 1063)



**NAME**

XARETVAL - XA Return Value

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1904'

**Length** \*

**Class** CLASS

**Sprcls** SCALAR - Scalar Object

Return values used in an XA environment to convey XA return code information to a requester. Valid values are described in *DTP: The XA+ Specification*. All errors that begin with XAER imply Severity Code 8 and are chain terminating errors.

clsvar	NIL	
insvar	INSTANCE VARIABLES	
length	8	
class	X'1904'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	32
	ENUVAL	0 - XA_OK
	ENUVAL	3 - XA_RDONLY
	ENUVAL	4 - XA_RETRY
	ENUVAL	5 - XA_HEURMIX
	ENUVAL	6 - XA_HEURRB
	ENUVAL	7 - XA_HEURCOM
	ENUVAL	8 - XA_HEURHAZ
	ENUVAL	9 - XA_NOMIGRATE
	ENUVAL	10 - XA_RETRY_COMMFAIL
	ENUVAL	13 - XA_TWOPHASE
	ENUVAL	99 - XA_LCSNOTSUPP
	ENUVAL	100 - XA_RBROLLBACK
	ENUVAL	101 - XA_RBCOMMFAIL
	ENUVAL	102 - XA_RBDEADLOCK
	ENUVAL	103 - XA_RBINTEGRITY
	ENUVAL	106 - XA_RBTIMEOUT
	ENUVAL	108 - XA_NODISSOCIATE
	ENUVAL	-3 - XAER_RMERR
	ENUVAL	-4 - XAER_NOTA
	ENUVAL	-5 - XAER_INVALID
	ENUVAL	-6 - XAER_PROTO
	ENUVAL	-7 - XAER_RMFAIL
	ENUVAL	-8 - XAER_DUPID
	ENUVAL	-9 - XAER_OUTSIDE
	DFTVAL	0 - XA_OK
	OPTIONAL	

NOTE Required if the received command was related to XA support. For each XA request, only certain XARETVAL values are allowed to be returned.

---

<b>clscmd</b>	<b>NIL</b>
<b>inscmd</b>	<b>NIL</b>

---

**SEE ALSO**

**insvar** *SYNCCRD* (on page 913)  
**Semantic** *XAMGROV* (on page 1066)

NAME

XID — Global Transaction Identifier

DESCRIPTION (Semantic)

**Dictionary** QDDBASD  
**Codepoint** X'1801'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

The syntax and semantics of the Global Transaction identifier is defined by The Open Group *Distributed Transaction Processing (DTP) Reference Model*. The content and description of each instance variable that constitutes an XID are not described by DDM. Refer to the The Open Group *DTP: The XA Specification* for a description and semantics of each field. The Open Group XID format is as follows:

```
#define XIDDATASIZE 128          /* size in bytes */
struct xid_t {
    long formatid;              /* format identifier */
    long gtrid_length;          /* value 1-64 */
    long bqual_length;          /* value 1-64 */
    char data[XIDDATASIZE];     /* gtrid | bqual */
};
typedef struct xid_t XID;
```

A value of -1 in *formatid* means the XID is null. If the XID passed is null, the target server should assume the connection is not part of a Global Transaction.

clsvar	NIL	
insvar	CLASS INSTANCE VARIABLES	
bqual_length	INSTANCE_OF	BINDR - Binary Number Field
	NOTE	Contains a number between 1 and 64.
	NOTE	The content and description of this number is not described by DDM.
	LENGTH	32
	MINLVL	7
data	OPTIONAL	
	INSTANCE_OF	BYTSTRDR - Byte String
	NOTE	The content and description of this value is not described by DDM.
	MINLEN	1
	MAXLEN	128
formatid	MINLVL	7
	OPTIONAL	
	INSTANCE_OF	BINDR - Binary Number Field
LENGTH	LENGTH	32
	NOTE	A value of -1 in <i>formatid</i> means the XID is null.

	NOTE	If null, the <i>gtrid_length</i> , <i>bqual_length</i> , and data are not specified.
	NOTE	The content and description of this number is not described by DDM.
	MINLVL REQUIRED	7
<i>gtrid_length</i>	INSTANCE_OF NOTE NOTE	BINDR - Binary Number Field Contains a number between 1 and 64. The content and description of this number is not described by DDM.
	LENGTH MINLVL OPTIONAL	32 7
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

<b>insvar</b>	<i>PRPHRCLST</i> (on page 635) <i>SYNCCTL</i> (on page 915)
<b>Semantic</b>	<i>RDB</i> (on page 718) <i>SYNCPTMGR</i> (on page 939)

**NAME**

XIDCNT - XID List Count

**DESCRIPTION (Semantic)**

**Dictionary** QDDBASD

**Codepoint** X'1906'

**Length** \*

**Class** CLASS

**Sprcls** SCALAR - Scalar Object

XIDCNT is the number of XID entries in the list of prepared or heuristically completed XIDs (PHPHRCLST).

<b>clsvar</b>	<b>NIL</b>	
<b>insvar</b>	<b>CLASS INSTANCE VARIABLES</b>	
length	6	
class	X'1906'	
value	INSTANCE_OF	BINDR - Binary Number Field
	LENGTH	16
<b>clscmd</b>	<b>NIL</b>	
<b>inscmd</b>	<b>NIL</b>	

**SEE ALSO**

**insvar** *PRPHRCLST* (on page 635)

NAME

XIDSHR - Share Recoverable Resources

DESCRIPTION (Semantic)

**Dictionary** QDDBASD  
**Codepoint** X'1802'  
**Length** \*  
**Class** CLASS  
**Sprcls** SCALAR - Scalar Object

An indicator that the XA application is designed to allow for partial or complete sharing of recoverable resources and locks across the Global Transaction identified by the XID. If set to 1, data is to be shared among agents with the same *gtrid*, but different *bqual* values must send XIDSHR with a value of 1. Refer to The Open Group *Distributed Transaction Processing (DTP) Reference Model* for a description of Global Transactions. If the target server has a tightly-coupled relationship (entire XID is equal) to a Global Transaction, DTP requires the RDB to prevent resource deadlocks from other processes participating in the Global Transaction. This is called *partial sharing* of resources across a Global Transaction. If the target server has a loosely-coupled relationship (*gtrid* is equal) to a Global Transaction, an application may indicate to the RDB to prevent resource deadlocks from all processes participating in the Global Transaction. This is called *complete sharing* of resources across a Global Transaction.

<b>clsvar</b>	NIL	
<b>insvar</b>	CLASS INSTANCE VARIABLES	
xidshr	INSTANCE_OF	BYTDR - An 8-bit Data Representation Value
	ENUVAL	X'00'
	NOTE	Partial Sharing - Share recoverable resources and locks when XID is equal.
	ENUVAL	X'01'
	NOTE	Complete Sharing - Share recoverable resources and locks when gtrid is equal.
xidshr	MINLVL	7
	DFTVAL	X'00'
<b>clscmd</b>	NIL	
<b>inscmd</b>	NIL	

SEE ALSO

**insvar** SYNCCTL (on page 915)  
**Semantic** QDDBASD (on page 651)  
RDB (on page 718)  
XAMGROV (on page 1066)

**NAME**

YMDBLKDATFMT — YMD with Blank Separator Date Format

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2454'**Length** \***Class** CODPNT

The YMD (Year, Month, Day) with Blank Separator Date Format (YMDBLKDATFMT) specifies that dates in the Structured Query Language (SQL) statements are in the following YMD date format:

yy mm dd

where the " " separator is a blank (with the Graphic Character Global Identifier (GCGID) SP01). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO****insvar** *STTDATFMT* (on page 894)**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)

**NAME**

YMDCMADATFMT — YMD with Comma Separator Date Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2455'

**Length** \*

**Class** CODPNT

The YMD (Year, Month, Day) with Comma Separator Date Format (YMDCMADATFMT) specifies that dates in the Structured Query Language (SQL) statements are in the following YMD date format:

*yy, mm, dd*

where the "," separator is a comma (with the Graphic Character Global Identifier (GCGID) SP08). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar** *STTDATFMT* (on page 894)

**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)



**NAME**

YMDHPNDATFMT — YMD with Hyphen Separator Date Format

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2456'**Length** \***Class** CODPNT

The YMD (Year, Month, Day) with Hyphen Separator Date Format (YMDHPNDATFMT) specifies that dates in the Structured Query Language (SQL) statements are in the following YMD date format:

yy-mm-dd

where the "-" separator is a hyphen (with the Graphic Character Global Identifier (GCGID) SP10). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO****insvar** *STTDATFMT* (on page 894)**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)

**NAME**

YMDPRDDATFMT — YMD with Period Separator Date Format

**DESCRIPTION (Semantic)**

**Dictionary** QDDRDBD

**Codepoint** X'2457'

**Length** \*

**Class** CODPNT

The YMD (Year, Month, Day) with Period Separator Date Format (YMDPRDDATFMT) specifies that dates in the Structured Query Language (SQL) statements are in the following YMD date format:

yy.mm.dd

where the "." separator is a period (with the Graphic Character Global Identifier (GCGID) SP11). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO**

**insvar** *STTDATFMT* (on page 894)

**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)

**NAME**

YMDSLHDATFMT — YMD with Slash Separator Date Format

**DESCRIPTION (Semantic)****Dictionary** QDDRDBD**Codepoint** X'2458'**Length** \***Class** CODPNT

The YMD (Year, Month, Day) with Slash Separator Date Format (YMDSLHDATFMT) specifies that dates in the Structured Query Language (SQL) statements are in the following YMD date format:

yy/mm/dd

where the "/" separator is a slash (with the Graphic Character Global Identifier (GCGID) SP12). For more information on GCGID, see the *Character Data Representation Architecture Registry* (SC09-1391, IBM).

**SEE ALSO****insvar** *STTDATFMT* (on page 894)**Semantic** *CODPNT* (on page 233)  
*QDDRDBD* (on page 657)



## Codepoints (Sorted by Term Name)

The following table lists the codepoints, sorted by term name:

<b>Term Name</b>	<b>Codepoint (Hex)</b>	<b>Reference</b>
ABNUOWRM	220D	<i>ABNUOWRM</i> (on page 39)
ACCDMG	003E	<i>ACCDMG</i> (on page 41)
ACCRDB	2001	<i>ACCRDB</i> (on page 42)
ACCRDBRM	2201	<i>ACCRDBRM</i> (on page 48)
ACCSEC	106D	<i>ACCSEC</i> (on page 52)
ACCSECRD	14AC	<i>ACCSECRD</i> (on page 58)
AGENT	1403	<i>AGENT</i> (on page 61)
AGNPRMRM	1232	<i>AGNPRMRM</i> (on page 65)
ARMCORR	2161	<i>ARMCORR</i> (on page 100)
ARRAY	004B	<i>ARRAY</i> (on page 101)
ASSOCIATION	0001	<i>ASSOCIATION</i> (on page 103)
ATMIND	2159	<i>ATMIND</i> (on page 104)
ATTLST	0046	<i>ATTLST</i> (on page 105)
BGNATMCHN	1803	<i>BGNATMCHN</i> (on page 107)
BGNBND	2002	<i>BGNBND</i> (on page 110)
BGNBNDRM	2208	<i>BGNBNDRM</i> (on page 117)
BIN	0003	<i>BIN</i> (on page 118)
BINDR	0042	<i>BINDR</i> (on page 120)
BITDR	0004	<i>BITDR</i> (on page 123)
BITSTRDR	0005	<i>BITSTRDR</i> (on page 124)
BNDCHKEXS	211B	<i>BNDCHKEXS</i> (on page 125)
BNDCHKONL	2421	<i>BNDCHKONL</i> (on page 126)
BNDCRTCTL	211D	<i>BNDCRTCTL</i> (on page 127)
BNDERRALW	2423	<i>BNDERRALW</i> (on page 128)
BNDEXPOPT	2130	<i>BNDEXPOPT</i> (on page 129)
BNDEXSOPT	241D	<i>BNDEXSOPT</i> (on page 130)
BNDEXSRQR	241C	<i>BNDEXSRQR</i> (on page 131)
BNDNERALW	2422	<i>BNDNERALW</i> (on page 132)
BNDOPT	2405	<i>BNDOPT</i> (on page 133)
BNDOPTNM	2144	<i>BNDOPTNM</i> (on page 134)

<b>Term Name</b>	<b>Codepoint (Hex)</b>	<b>Reference</b>
BNDOPTVL	2145	<i>BNDOPTVL</i> (on page 135)
BNDSQLSTT	2004	<i>BNDSQLSTT</i> (on page 136)
BNDSTTASM	2126	<i>BNDSTTASM</i> (on page 141)
BOOLEAN	0006	<i>BOOLEAN</i> (on page 142)
BUFINSIND	215C	<i>BUFINSIND</i> (on page 143)
BYTDR	0043	<i>BYTDR</i> (on page 145)
BYTSTRDR	0044	<i>BYTSTRDR</i> (on page 146)
CCSIDDBC	119D	<i>CCSIDDBC</i> (on page 148)
CCSIDMBC	119E	<i>CCSIDMBC</i> (on page 149)
CCSIDMGR	14CC	<i>CCSIDMGR</i> (on page 150)
CCSIDSBC	119C	<i>CCSIDSBC</i> (on page 154)
CHRDR	0008	<i>CHRDR</i> (on page 155)
CHRSTRDR	0009	<i>CHRSTRDR</i> (on page 156)
CLASS	000A	<i>CLASS</i> (on page 158)
CLSQRV	2005	<i>CLSQRV</i> (on page 165)
CMDATHRM	121C	<i>CMDATHRM</i> (on page 171)
CMDCHKRM	1254	<i>CMDCHKRM</i> (on page 173)
CMDCMPRM	124B	<i>CMDCMPRM</i> (on page 175)
CMDNSPRM	1250	<i>CMDNSPRM</i> (on page 176)
CMDSRCID	2107	<i>CMDSRCID</i> (on page 178)
CMDTRG	0041	<i>CMDTRG</i> (on page 179)
CMDVLTRM	221D	<i>CMDVLTRM</i> (on page 181)
CMMRQSRM	2225	<i>CMMRQSRM</i> (on page 182)
CMMTYP	2143	<i>CMMTYP</i> (on page 183)
CMNAPPC	1444	<i>CMNAPPC</i> (on page 184)
CMNMGR	1408	<i>CMNMGR</i> (on page 196)
CMNSYNCPT	147C	<i>CMNSYNCPT</i> (on page 202)
CMNTCPIP	1474	<i>CMNTCPIP</i> (on page 214)
CNNTKN	1070	<i>CNNTKN</i> (on page 220)
CNSVAL	000B	<i>CNSVAL</i> (on page 221)
CNTQRY	2006	<i>CNTQRY</i> (on page 222)
CODPNT	000C	<i>CODPNT</i> (on page 233)

*Codepoints (Sorted by Term Name)*

<b>Term Name</b>	<b>Codepoint (Hex)</b>	<b>Reference</b>
CODPNTDR	0064	<i>CODPNTDR</i> (on page 235)
COLLECTION	000D	<i>COLLECTION</i> (on page 238)
COMMAND	000E	<i>COMMAND</i> (on page 240)
CONSTANT	0050	<i>CONSTANT</i> (on page 244)
CRRTKN	2135	<i>CRRTKN</i> (on page 246)
CSTBITS	2433	<i>CSTBITS</i> (on page 247)
CSTMBCS	2435	<i>CSTMBCS</i> (on page 248)
CSTSBCS	2434	<i>CSTSBCS</i> (on page 249)
CSTSYSDFT	2432	<i>CSTSYSDFT</i> (on page 250)
DATA	003C	<i>DATA</i> (on page 251)
DCTIND	1450	<i>DCTIND</i> (on page 258)
DCTINDEN	1451	<i>DCTINDEN</i> (on page 259)
DECDELMA	243D	<i>DECDELMA</i> (on page 264)
DECDELPRD	243C	<i>DECDELPRD</i> (on page 265)
DECPRC	2106	<i>DECPRC</i> (on page 266)
DEFINITION	0048	<i>DEFINITION</i> (on page 267)
DEFLST	0047	<i>DEFLST</i> (on page 268)
DEPERLCD	119B	<i>DEPERLCD</i> (on page 270)
DFTDATFMT	2400	<i>DFTDATFMT</i> (on page 272)
DFTPKG	241E	<i>DFTPKG</i> (on page 273)
DFTRBCOL	2128	<i>DFTRBCOL</i> (on page 274)
DFTTIMFMT	2401	<i>DFTTIMFMT</i> (on page 275)
DFTVAL	0011	<i>DFTVAL</i> (on page 276)
DGRIOPRL	212F	<i>DGRIOPRL</i> (on page 279)
DIAGLVL	2160	<i>DIAGLVL</i> (on page 284)
DICTIONARY	1458	<i>DICTIONARY</i> (on page 286)
DMYBLKDATFMT	2402	<i>DMYBLKDATFMT</i> (on page 288)
DMYCMADATFMT	2403	<i>DMYCMADATFMT</i> (on page 289)
DMYHPNDATFMT	2404	<i>DMYHPNDATFMT</i> (on page 290)
DMYPRDDATFMT	2406	<i>DMYPRDDATFMT</i> (on page 291)
DMYSLHDATFMT	2409	<i>DMYSLHDATFMT</i> (on page 292)
DRPPKG	2007	<i>DRPPKG</i> (on page 293)

<b>Term Name</b>	<b>Codepoint (Hex)</b>	<b>Reference</b>
DSCERRCD	2101	<i>DSCERRCD</i> (on page 296)
DSCINVRM	220A	<i>DSCINVRM</i> (on page 298)
DSCRDBTBL	2012	<i>DSCRDBTBL</i> (on page 300)
DSCSQLSTT	2008	<i>DSCSQLSTT</i> (on page 304)
DSSFMT	140D	<i>DSSFMT</i> (on page 318)
DTAMCHRM	220E	<i>DTAMCHRM</i> (on page 320)
DUPQRYOK	210B	<i>DUPQRYOK</i> (on page 322)
ELMCLS	004D	<i>ELMCLS</i> (on page 330)
ENCALG	1909	<i>ENCALG</i> (on page 331)
ENCKEYLEN	190A	<i>ENCKEYLEN</i> (on page 332)
ENDATMCHN	1804	<i>ENDATMCHN</i> (on page 333)
ENDBND	2009	<i>ENDBND</i> (on page 336)
ENDCHNTYP	1902	<i>ENDCHNTYP</i> (on page 341)
ENDQRYRM	220B	<i>ENDQRYRM</i> (on page 342)
ENDUOWRM	220C	<i>ENDUOWRM</i> (on page 343)
ENUCLS	0040	<i>ENUCLS</i> (on page 345)
ENULEN	0015	<i>ENULEN</i> (on page 346)
ENUVAL	0016	<i>ENUVAL</i> (on page 347)
ERROR	0017	<i>ERROR</i> (on page 350)
ETIME	1901	<i>ETIME</i> (on page 352)
EURDATFMT	242B	<i>EURDATFMT</i> (on page 353)
EURTIMFMT	2430	<i>EURTIMFMT</i> (on page 354)
EXCSAT	1041	<i>EXCSAT</i> (on page 363)
EXCSATRD	1443	<i>EXCSATRD</i> (on page 369)
EXCSQLIMM	200A	<i>EXCSQLIMM</i> (on page 371)
EXCSQLSET	2014	<i>EXCSQLSET</i> (on page 377)
EXCSQLSTT	200B	<i>EXCSQLSTT</i> (on page 381)
EXPALL	243B	<i>EXPALL</i> (on page 393)
EXPNON	243A	<i>EXPNON</i> (on page 394)
EXPREOPT	2459	<i>EXPREOPT</i> (on page 395)
EXPYES	240A	<i>EXPYES</i> (on page 396)
EXTDTA	146C	<i>EXTDTA</i> (on page 397)



*Codepoints (Sorted by Term Name)*

<b>Term Name</b>	<b>Codepoint (Hex)</b>	<b>Reference</b>
EXTNAM	115E	<i>EXTNAM</i> (on page 402)
FALSE	0018	<i>FALSE</i> (on page 403)
FDODSC	0010	<i>FDODSC</i> (on page 406)
FDODSCOFF	2118	<i>FDODSCOFF</i> (on page 407)
FDODTA	147A	<i>FDODTA</i> (on page 408)
FDODTAOFF	2119	<i>FDODTAOFF</i> (on page 409)
FDOEXT	147B	<i>FDOEXT</i> (on page 410)
FDOOBJ	1480	<i>FDOOBJ</i> (on page 411)
FDOOFF	147D	<i>FDOOFF</i> (on page 412)
FDOPRMOFF	212B	<i>FDOPRMOFF</i> (on page 413)
FDOTRPOFF	212A	<i>FDOTRPOFF</i> (on page 414)
FIELD	006A	<i>FIELD</i> (on page 415)
FIXROWPRC	2418	<i>FIXROWPRC</i> (on page 417)
FORGET	1186	<i>FORGET</i> (on page 423)
FRCFIXROW	2410	<i>FRCFIXROW</i> (on page 424)
HELP	0019	<i>HELP</i> (on page 425)
HEXDR	001A	<i>HEXDR</i> (on page 428)
HEXSTRDR	001B	<i>HEXSTRDR</i> (on page 429)
HMSBLKTIMFMT	240C	<i>HMSBLKTIMFMT</i> (on page 431)
HMSCLNTIMFMT	240D	<i>HMSCLNTIMFMT</i> (on page 432)
HMSCMATIMFMT	2416	<i>HMSCMATIMFMT</i> (on page 433)
HMSPRDTIMFMT	2428	<i>HMSPRDTIMFMT</i> (on page 434)
IGNORABLE	001C	<i>IGNORABLE</i> (on page 435)
INFO	001E	<i>INFO</i> (on page 436)
INHERITED	0049	<i>INHERITED</i> (on page 442)
INSTANCE_OF	005D	<i>INSTANCE_OF</i> (on page 444)
INTRDBRQS	2003	<i>INTRDBRQS</i> (on page 445)
INTTKNRM	2210	<i>INTTKNRM</i> (on page 448)
IPADDR	11E8	<i>IPADDR</i> (on page 450)
ISODATFMT	2429	<i>ISODATFMT</i> (on page 451)
ISOLVLALL	2443	<i>ISOLVLALL</i> (on page 452)
ISOLVLCHG	2441	<i>ISOLVLCHG</i> (on page 453)

<b>Term Name</b>	<b>Codepoint (Hex)</b>	<b>Reference</b>
ISOLVLCS	2442	<i>ISOLVLCS</i> (on page 454)
ISOLVLNC	2445	<i>ISOLVLNC</i> (on page 455)
ISOLVLR	2444	<i>ISOLVLR</i> (on page 456)
ISOTIMFMT	242E	<i>ISOTIMFMT</i> (on page 457)
JISDATFMT	242C	<i>JISDATFMT</i> (on page 458)
JISTIMFMT	2431	<i>JISTIMFMT</i> (on page 459)
JULBLKDATFMT	242D	<i>JULBLKDATFMT</i> (on page 460)
JULCMADATFMT	243F	<i>JULCMADATFMT</i> (on page 461)
JULHPNDATFMT	2440	<i>JULHPNDATFMT</i> (on page 462)
JULPRDDATFMT	2446	<i>JULPRDDATFMT</i> (on page 463)
JULSLHDATFMT	2447	<i>JULSLHDATFMT</i> (on page 464)
KERSECPPL	1C02	<i>KERSECPPL</i> (on page 470)
LENGTH	001F	<i>LENGTH</i> (on page 472)
LMTBLKPRC	2417	<i>LMTBLKPRC</i> (on page 475)
LOCDATFMT	2448	<i>LOCDATFMT</i> (on page 482)
LOCTIMFMT	2449	<i>LOCTIMFMT</i> (on page 483)
LOGNAME	1184	<i>LOGNAME</i> (on page 484)
LOGTSTMP	1185	<i>LOGTSTMP</i> (on page 485)
MANAGER	1456	<i>MANAGER</i> (on page 491)
MAXBLKEXT	2141	<i>MAXBLKEXT</i> (on page 494)
MAXLEN	0021	<i>MAXLEN</i> (on page 495)
MAXRSLCNT	2140	<i>MAXRSLCNT</i> (on page 497)
MAXSCTNBR	2127	<i>MAXSCTNBR</i> (on page 498)
MAXVAL	0022	<i>MAXVAL</i> (on page 499)
MDYBLKDATFMT	244A	<i>MDYBLKDATFMT</i> (on page 500)
MDYCMADATFMT	244B	<i>MDYCMADATFMT</i> (on page 501)
MDYHPNDATFMT	2451	<i>MDYHPNDATFMT</i> (on page 502)
MDYPRDDATFMT	2452	<i>MDYPRDDATFMT</i> (on page 503)
MDYSLHDATFMT	2453	<i>MDYSLHDATFMT</i> (on page 504)
MGRDEPRM	1218	<i>MGRDEPRM</i> (on page 505)
MGRLVL	1442	<i>MGRLVL</i> (on page 506)
MGRLVLLS	1404	<i>MGRLVLLS</i> (on page 508)

*Codepoints (Sorted by Term Name)*

<b>Term Name</b>	<b>Codepoint (Hex)</b>	<b>Reference</b>
MGRLVLN	1473	<i>MGRLVLN</i> (on page 509)
MGRLVLOVR	1C03	<i>MGRLVLOVR</i> (on page 511)
MGRLVLRM	1210	<i>MGRLVLRM</i> (on page 512)
MGRNAM	1452	<i>MGRNAM</i> (on page 513)
MINLEN	0025	<i>MINLEN</i> (on page 517)
MINLVL	0002	<i>MINLVL</i> (on page 518)
MINVAL	0026	<i>MINVAL</i> (on page 519)
MONITOR	1900	<i>MONITOR</i> (on page 521)
MONITORRD	1C00	<i>MONITORRD</i> (on page 523)
MTLEXC	0067	<i>MTLEXC</i> (on page 524)
MTLINC	00A7	<i>MTLINC</i> (on page 525)
NAMDR	0066	<i>NAMDR</i> (on page 526)
NAME	0027	<i>NAME</i> (on page 527)
NAMSYMDR	0061	<i>NAMSYMDR</i> (on page 528)
NBRROW	213A	<i>NBRROW</i> (on page 530)
NEWPASSWORD	11DE	<i>NEWPASSWORD</i> (on page 531)
NIL	002A	<i>NIL</i> (on page 532)
NOTE	0014	<i>NOTE</i> (on page 533)
NUMBER	002B	<i>NUMBER</i> (on page 534)
OBJDSS	1429	<i>OBJDSS</i> (on page 536)
OBJECT	002C	<i>OBJECT</i> (on page 540)
OBJNSPRM	1253	<i>OBJNSPRM</i> (on page 542)
OPNQFLRM	2212	<i>OPNQFLRM</i> (on page 554)
OPNQRY	200C	<i>OPNQRY</i> (on page 555)
OPNQRYRM	2205	<i>OPNQRYRM</i> (on page 566)
OPTIONAL	002D	<i>OPTIONAL</i> (on page 568)
ORDCOL	004C	<i>ORDCOL</i> (on page 569)
OUTEXP	2111	<i>OUTEXP</i> (on page 571)
OUTOVR	2415	<i>OUTOVR</i> (on page 572)
OUTOVROPT	2147	<i>OUTOVROPT</i> (on page 576)
PASSWORD	11A1	<i>PASSWORD</i> (on page 578)
PKGATHKP	2425	<i>PKGATHKP</i> (on page 579)

<b>Term Name</b>	<b>Codepoint (Hex)</b>	<b>Reference</b>
PKGATHOPT	211E	<i>PKGATHOPT</i> (on page 580)
PKGATHRUL	213F	<i>PKGATHRUL</i> (on page 581)
PKGATHRVK	2424	<i>PKGATHRVK</i> (on page 584)
PKGBNARM	2206	<i>PKGBNARM</i> (on page 585)
PKGBPARM	2209	<i>PKGBPARM</i> (on page 586)
PKGCNSTKN	210D	<i>PKGCNSTKN</i> (on page 588)
PKGDFTCC	119A	<i>PKGDFTCC</i> (on page 589)
PKGDFTCST	2125	<i>PKGDFTCST</i> (on page 590)
PKGID	2109	<i>PKGID</i> (on page 591)
PKGISOLVL	2124	<i>PKGISOLVL</i> (on page 592)
PKGNAME	210A	<i>PKGNAME</i> (on page 594)
PKGNAMECSN	2113	<i>PKGNAMECSN</i> (on page 596)
PKGNAMECT	2112	<i>PKGNAMECT</i> (on page 598)
PKGOWNID	2131	<i>PKGOWNID</i> (on page 600)
PKGRPLALW	241F	<i>PKGRPLALW</i> (on page 602)
PKGRPLNA	2420	<i>PKGRPLNA</i> (on page 603)
PKGRPLOPT	211C	<i>PKGRPLOPT</i> (on page 604)
PKGRPLVRS	212D	<i>PKGRPLVRS</i> (on page 605)
PKGSN	210C	<i>PKGSN</i> (on page 606)
PKGSNLST	2139	<i>PKGSNLST</i> (on page 607)
PKTOBJ	1C04	<i>PKTOBJ</i> (on page 608)
PLGINCNT	190F	<i>PLGINCNT</i> (on page 610)
PLGINID	190D	<i>PLGINID</i> (on page 611)
PLGINLSE	1910	<i>PLGINLSE</i> (on page 612)
PLGINLST	191E	<i>PLGINLST</i> (on page 613)
PLGINNM	191C	<i>PLGINNM</i> (on page 614)
PLGINPPL	1911	<i>PLGINPPL</i> (on page 619)
PRCCNVCD	113F	<i>PRCCNVCD</i> (on page 621)
PRCCNVRM	1245	<i>PRCCNVRM</i> (on page 625)
PRCNAM	2138	<i>PRCNAM</i> (on page 627)
PRDDTA	2104	<i>PRDDTA</i> (on page 630)
PRDID	112E	<i>PRDID</i> (on page 631)

*Codepoints (Sorted by Term Name)*

<b>Term Name</b>	<b>Codepoint (Hex)</b>	<b>Reference</b>
PRMDMG	002E	<i>PRMDMG</i> (on page 632)
PRMNSPRM	1251	<i>PRMNSPRM</i> (on page 633)
PRPHRCLST	1905	<i>PRPHRCLST</i> (on page 635)
PRPSQLSTT	200D	<i>PRPSQLSTT</i> (on page 636)
PRPSTTKP	2167	<i>PRPSTTKP</i> (on page 642)
QLFATT	0007	<i>QLFATT</i> (on page 664)
QRYATTSCR	2149	<i>QRYATTSCR</i> (on page 665)
QRYATTSET	214A	<i>QRYATTSET</i> (on page 666)
QRYATTSNS	2157	<i>QRYATTSNS</i> (on page 667)
QRYATTUPD	2150	<i>QRYATTUPD</i> (on page 669)
QRYBLKCTL	2132	<i>QRYBLKCTL</i> (on page 672)
QRYBLKFCT	215F	<i>QRYBLKFCT</i> (on page 675)
QRYBLKRST	2154	<i>QRYBLKRST</i> (on page 677)
QRYBLKSZ	2114	<i>QRYBLKSZ</i> (on page 678)
QRYBLKTYP	2133	<i>QRYBLKTYP</i> (on page 679)
QRYCLSIMP	215D	<i>QRYCLSIMP</i> (on page 680)
QRYCLSRLS	215E	<i>QRYCLSRLS</i> (on page 682)
QRYDSC	241A	<i>QRYDSC</i> (on page 685)
QRYDTA	241B	<i>QRYDTA</i> (on page 686)
QRYINSID	215B	<i>QRYINSID</i> (on page 688)
QRYNOPRM	2202	<i>QRYNOPRM</i> (on page 690)
QRYPOPRM	220F	<i>QRYPOPRM</i> (on page 691)
QRYPRCTYP	2102	<i>QRYPRCTYP</i> (on page 692)
QRYROWNBR	213D	<i>QRYROWNBR</i> (on page 694)
QRYROWSET	2156	<i>QRYROWSET</i> (on page 697)
QRYROWSNS	2153	<i>QRYROWSNS</i> (on page 701)
QRYRTNDTA	2155	<i>QRYRTNDTA</i> (on page 702)
QRYSCRORN	2152	<i>QRYSCRORN</i> (on page 710)
RDB	240F	<i>RDB</i> (on page 718)
RDBACCCL	210F	<i>RDBACCCL</i> (on page 723)
RDBACCRM	2207	<i>RDBACCRM</i> (on page 724)
RDBAFLRM	221A	<i>RDBAFLRM</i> (on page 725)

<b>Term Name</b>	<b>Codepoint (Hex)</b>	<b>Reference</b>
RDBALWUPD	211A	<i>RDBALWUPD</i> (on page 726)
RDBATHRM	2203	<i>RDBATHRM</i> (on page 727)
RDBCMM	200E	<i>RDBCMM</i> (on page 728)
RDBCMTOK	2105	<i>RDBCMTOK</i> (on page 731)
RDBCOLID	2108	<i>RDBCOLID</i> (on page 732)
RDBINTTKN	2103	<i>RDBINTTKN</i> (on page 733)
RDBNACRM	2204	<i>RDBNACRM</i> (on page 734)
RDBNAM	2110	<i>RDBNAM</i> (on page 736)
RDBNFNRM	2211	<i>RDBNFNRM</i> (on page 739)
RDBRLLBCK	200F	<i>RDBRLLBCK</i> (on page 745)
RDBRLSCMM	2438	<i>RDBRLSCMM</i> (on page 748)
RDBRLSCNV	2439	<i>RDBRLSCNV</i> (on page 749)
RDBRLSOPT	2129	<i>RDBRLSOPT</i> (on page 750)
RDBUPDRM	2218	<i>RDBUPDRM</i> (on page 751)
REBIND	2010	<i>REBIND</i> (on page 753)
REPEATABLE	0031	<i>REPEATABLE</i> (on page 758)
REQUIRED	0032	<i>REQUIRED</i> (on page 761)
RESERVED	0033	<i>RESERVED</i> (on page 762)
RESPKTSZ	1908	<i>RESPKTSZ</i> (on page 763)
RLSCONV	119F	<i>RLSCONV</i> (on page 765)
RPYDSS	1436	<i>RPYDSS</i> (on page 766)
RPYMSG	1437	<i>RPYMSG</i> (on page 770)
RQSCRR	1438	<i>RQSCRR</i> (on page 772)
RQSDSS	1439	<i>RQSDSS</i> (on page 774)
RSCLMTRM	1233	<i>RSCLMTRM</i> (on page 778)
RSCNAM	112D	<i>RSCNAM</i> (on page 781)
RSCTYP	111F	<i>RSCTYP</i> (on page 782)
RSLSETFLG	2142	<i>RSLSETFLG</i> (on page 783)
RSLSETRM	2219	<i>RSLSETRM</i> (on page 785)
RSNCOD	1127	<i>RSNCOD</i> (on page 786)
RSYNMGR	14C1	<i>RSYNMGR</i> (on page 787)
RSYNCTYP	11EA	<i>RSYNCTYP</i> (on page 790)

*Codepoints (Sorted by Term Name)*

<b>Term Name</b>	<b>Codepoint (Hex)</b>	<b>Reference</b>
RTNEXTDTA	2148	<i>RTNEXTDTA</i> (on page 792)
RTNSETSTT	210E	<i>RTNSETSTT</i> (on page 794)
RTNSQLDA	2116	<i>RTNSQLDA</i> (on page 795)
SCALAR	0034	<i>SCALAR</i> (on page 796)
SCLDTALEN	0100	<i>SCLDTALEN</i> (on page 799)
SECCHK	106E	<i>SECCHK</i> (on page 800)
SECCHKCD	11A4	<i>SECCHKCD</i> (on page 806)
SECCHKRM	1219	<i>SECCHKRM</i> (on page 809)
SECMEC	11A2	<i>SECMEC</i> (on page 811)
SECMGR	1440	<i>SECMGR</i> (on page 814)
SECMGRNM	1196	<i>SECMGRNM</i> (on page 818)
SECTKN	11DC	<i>SECTKN</i> (on page 820)
SECTKNOVR	190B	<i>SECTKNOVR</i> (on page 822)
SERVER	1448	<i>SERVER</i> (on page 824)
SESDMG	003F	<i>SESDMG</i> (on page 825)
SEVERE	003A	<i>SEVERE</i> (on page 826)
SNAADDR	11E9	<i>SNAADDR</i> (on page 827)
SNDPKT	1805	<i>SNDPKT</i> (on page 829)
SPCVL	0036	<i>SPCVL</i> (on page 831)
SPRCLS	0037	<i>SPRCLS</i> (on page 832)
SPVNAM	115D	<i>SPVNAM</i> (on page 834)
SQLAM	2407	<i>SQLAM</i> (on page 847)
SQLATTR	2450	<i>SQLATTR</i> (on page 854)
SQLCARD	2408	<i>SQLCARD</i> (on page 855)
SQLCINRD	240B	<i>SQLCINRD</i> (on page 857)
SQLCSRHLD	211F	<i>SQLCSRHLD</i> (on page 858)
SQLDARD	2411	<i>SQLDARD</i> (on page 859)
SQLDTA	2412	<i>SQLDTA</i> (on page 860)
SQLDTARD	2413	<i>SQLDTARD</i> (on page 862)
SQLERRRM	2213	<i>SQLERRRM</i> (on page 864)
SQLOBJNAM	243E	<i>SQLOBJNAM</i> (on page 866)
SQLRSLRD	240E	<i>SQLRSLRD</i> (on page 867)

<b>Term Name</b>	<b>Codepoint (Hex)</b>	<b>Reference</b>
SQLSTT	2414	<i>SQLSTT</i> (on page 868)
SQLSTTNBR	2117	<i>SQLSTTNBR</i> (on page 870)
SQLSTTVRB	2419	<i>SQLSTTVRB</i> (on page 871)
SRVCLSNM	1147	<i>SRVCLSNM</i> (on page 872)
SRVDGN	1153	<i>SRVDGN</i> (on page 873)
SRVLCNT	244C	<i>SRVLCNT</i> (on page 875)
SRVLSRV	244D	<i>SRVLSRV</i> (on page 876)
SRVLST	244E	<i>SRVLST</i> (on page 878)
SRVNAM	116D	<i>SRVNAM</i> (on page 880)
SRVPRTY	244F	<i>SRVPRTY</i> (on page 883)
SRVRLSLV	115A	<i>SRVRLSLV</i> (on page 884)
STGLMT	1409	<i>STGLMT</i> (on page 885)
STRDELAP	2426	<i>STRDELAP</i> (on page 886)
STRDELDQ	2427	<i>STRDELDQ</i> (on page 887)
STRING	0038	<i>STRING</i> (on page 888)
STTASMEUI	2437	<i>STTASMEUI</i> (on page 893)
STTDATFMT	2122	<i>STTDATFMT</i> (on page 894)
STTDECDEL	2121	<i>STTDECDEL</i> (on page 897)
STTSCCCLS	2436	<i>STTSCCCLS</i> (on page 898)
STTSTRDEL	2120	<i>STTSTRDEL</i> (on page 899)
STTTIMFMT	2123	<i>STTTIMFMT</i> (on page 900)
SUPERVISOR	143C	<i>SUPERVISOR</i> (on page 908)
SVCERRNO	11B4	<i>SVCERRNO</i> (on page 910)
SVRCOD	1149	<i>SVRCOD</i> (on page 911)
SYNCCRD	1248	<i>SYNCCRD</i> (on page 913)
SYNCCTL	1055	<i>SYNCCTL</i> (on page 915)
SYNCLOG	106F	<i>SYNCLOG</i> (on page 922)
SYNCPTMGR	14C0	<i>SYNCPTMGR</i> (on page 939)
SYNCRRD	126D	<i>SYNCRRD</i> (on page 981)
SYNCRSY	1069	<i>SYNCRSY</i> (on page 982)
SYNCTYPE	1187	<i>SYNCTYPE</i> (on page 984)
SYNERLCD	114A	<i>SYNERLCD</i> (on page 985)



*Codepoints (Sorted by Term Name)*

<b>Term Name</b>	<b>Codepoint (Hex)</b>	<b>Reference</b>
SYNTAXRM	124C	<i>SYNTAXRM</i> (on page 989)
TCPHOST	11DD	<i>TCPHOST</i> (on page 999)
TCPPTHOST	1912	<i>TCPPTHOST</i> (on page 1006)
TEXT	114B	<i>TEXT</i> (on page 1018)
TITLE	0045	<i>TITLE</i> (on page 1020)
TRGNSPRM	125F	<i>TRGNSPRM</i> (on page 1022)
TRUE	003B	<i>TRUE</i> (on page 1024)
TYPDEF	0029	<i>TYPDEF</i> (on page 1025)
TYPDEFNAM	002F	<i>TYPDEFNAM</i> (on page 1027)
TYPDEFOVR	0035	<i>TYPDEFOVR</i> (on page 1030)
TYPFMLNM	0030	<i>TYPFMLNM</i> (on page 1033)
TYPSQLDA	2146	<i>TYPSQLDA</i> (on page 1034)
UNORDERED	14B1	<i>UNORDERED</i> (on page 1036)
UOWDSP	2115	<i>UOWDSP</i> (on page 1037)
UOWID	11AA	<i>UOWID</i> (on page 1038)
UOWSTATE	11AC	<i>UOWSTATE</i> (on page 1040)
USADATFMT	242A	<i>USADATFMT</i> (on page 1041)
USATIMFMT	242F	<i>USATIMFMT</i> (on page 1042)
USRID	11A0	<i>USRID</i> (on page 1044)
VALNSPRM	1252	<i>VALNSPRM</i> (on page 1057)
VRSNAM	1144	<i>VRSNAM</i> (on page 1059)
WARNING	003D	<i>WARNING</i> (on page 1060)
XAFLAGS	1903	<i>XAFLAGS</i> (on page 1061)
XAMGR	1C01	<i>XAMGR</i> (on page 1063)
XARETVAL	1904	<i>XARETVAL</i> (on page 1107)
XID	1801	<i>XID</i> (on page 1109)
XIDCNT	1906	<i>XIDCNT</i> (on page 1111)
XIDSHR	1802	<i>XIDSHR</i> (on page 1112)
YMDBLKDATFMT	2454	<i>YMDBLKDATFMT</i> (on page 1113)
YMDCMADATFMT	2455	<i>YMDCMADATFMT</i> (on page 1114)
YMDHPNDATFMT	2456	<i>YMDHPNDATFMT</i> (on page 1115)
YMDPRDDATFMT	2457	<i>YMDPRDDATFMT</i> (on page 1116)
YMDSLHDATFMT	2458	<i>YMDSLHDATFMT</i> (on page 1117)



## Term Names (Sorted by Codepoints)

The following table lists the term name, sorted by codepoints:

<b>Codepoint (Hex)</b>	<b>Term Name</b>	<b>Reference</b>
0001	ASSOCIATION	<i>ASSOCIATION</i> (on page 103)
0002	MINLVL	<i>MINLVL</i> (on page 518)
0003	BIN	<i>BIN</i> (on page 118)
0004	BITDR	<i>BITDR</i> (on page 123)
0005	BITSTRDR	<i>BITSTRDR</i> (on page 124)
0006	BOOLEAN	<i>BOOLEAN</i> (on page 142)
0007	QLFATT	<i>QLFATT</i> (on page 664)
0008	CHRDR	<i>CHRDR</i> (on page 155)
0009	CHRSTRDR	<i>CHRSTRDR</i> (on page 156)
000A	CLASS	<i>CLASS</i> (on page 158)
000B	CNSVAL	<i>CNSVAL</i> (on page 221)
000C	CODPNT	<i>CODPNT</i> (on page 233)
000D	COLLECTION	<i>COLLECTION</i> (on page 238)
000E	COMMAND	<i>COMMAND</i> (on page 240)
0010	FDODSC	<i>FDODSC</i> (on page 406)
0011	DFTVAL	<i>DFTVAL</i> (on page 276)
0014	NOTE	<i>NOTE</i> (on page 533)
0015	ENULEN	<i>ENULEN</i> (on page 346)
0016	ENUVAL	<i>ENUVAL</i> (on page 347)
0017	ERROR	<i>ERROR</i> (on page 350)
0018	FALSE	<i>FALSE</i> (on page 403)
0019	HELP	<i>HELP</i> (on page 425)
001A	HEXDR	<i>HEXDR</i> (on page 428)
001B	HEXSTRDR	<i>HEXSTRDR</i> (on page 429)
001C	IGNORABLE	<i>IGNORABLE</i> (on page 435)
001E	INFO	<i>INFO</i> (on page 436)
001F	LENGTH	<i>LENGTH</i> (on page 472)
0021	MAXLEN	<i>MAXLEN</i> (on page 495)
0022	MAXVAL	<i>MAXVAL</i> (on page 499)

Codepoint (Hex)	Term Name	Reference
0025	MINLEN	<i>MINLEN</i> (on page 517)
0026	MINVAL	<i>MINVAL</i> (on page 519)
0027	NAME	<i>NAME</i> (on page 527)
0029	TYPDEF	<i>TYPDEF</i> (on page 1025)
002A	NIL	<i>NIL</i> (on page 532)
002B	NUMBER	<i>NUMBER</i> (on page 534)
002C	OBJECT	<i>OBJECT</i> (on page 540)
002D	OPTIONAL	<i>OPTIONAL</i> (on page 568)
002E	PRMDMG	<i>PRMDMG</i> (on page 632)
002F	TYPDEFNAM	<i>TYPDEFNAM</i> (on page 1027)
0030	TYPFMLNM	<i>TYPFMLNM</i> (on page 1033)
0031	REPEATABLE	<i>REPEATABLE</i> (on page 758)
0032	REQUIRED	<i>REQUIRED</i> (on page 761)
0033	RESERVED	<i>RESERVED</i> (on page 762)
0034	SCALAR	<i>SCALAR</i> (on page 796)
0035	TYPDEFOVR	<i>TYPDEFOVR</i> (on page 1030)
0036	SPCVAL	<i>SPCVAL</i> (on page 831)
0037	SPRCLS	<i>SPRCLS</i> (on page 832)
0038	STRING	<i>STRING</i> (on page 888)
003A	SEVERE	<i>SEVERE</i> (on page 826)
003B	TRUE	<i>TRUE</i> (on page 1024)
003C	DATA	<i>DATA</i> (on page 251)
003D	WARNING	<i>WARNING</i> (on page 1060)
003E	ACCDMG	<i>ACCDMG</i> (on page 41)
003F	SESDMG	<i>SESDMG</i> (on page 825)
0040	ENUCLS	<i>ENUCLS</i> (on page 345)
0041	CMDTRG	<i>CMDTRG</i> (on page 179)
0042	BINDR	<i>BINDR</i> (on page 120)
0043	BYTDR	<i>BYTDR</i> (on page 145)
0044	BYTSTRDR	<i>BYTSTRDR</i> (on page 146)
0045	TITLE	<i>TITLE</i> (on page 1020)
0046	ATTLST	<i>ATTLST</i> (on page 105)

*Term Names (Sorted by Codepoints)*

<b>Codepoint (Hex)</b>	<b>Term Name</b>	<b>Reference</b>
0047	DEFLST	<i>DEFLST</i> (on page 268)
0048	DEFINITION	<i>DEFINITION</i> (on page 267)
0049	INHERITED	<i>INHERITED</i> (on page 442)
004B	ARRAY	<i>ARRAY</i> (on page 101)
004C	ORDCOL	<i>ORDCOL</i> (on page 569)
004D	ELMCLS	<i>ELMCLS</i> (on page 330)
0050	CONSTANT	<i>CONSTANT</i> (on page 244)
005D	INSTANCE_OF	<i>INSTANCE_OF</i> (on page 444)
0061	NAMSYMDR	<i>NAMSYMDR</i> (on page 528)
0064	CODPNTDR	<i>CODPNTDR</i> (on page 235)
0066	NAMDR	<i>NAMDR</i> (on page 526)
0067	MTLEXC	<i>MTLEXC</i> (on page 524)
006A	FIELD	<i>FIELD</i> (on page 415)
00A7	MTLINC	<i>MTLINC</i> (on page 525)
0100	SCLDTALEN	<i>SCLDTALEN</i> (on page 799)
1041	EXCSAT	<i>EXCSAT</i> (on page 363)
1055	SYNCCTL	<i>SYNCCTL</i> (on page 915)
1069	SYNCRSY	<i>SYNCRSY</i> (on page 982)
106D	ACCSEC	<i>ACCSEC</i> (on page 52)
106E	SECCHK	<i>SECCHK</i> (on page 800)
106F	SYNCLOG	<i>SYNCLOG</i> (on page 922)
1070	CNNTKN	<i>CNNTKN</i> (on page 220)
111F	RSCTYP	<i>RSCTYP</i> (on page 782)
1127	RSNCOD	<i>RSNCOD</i> (on page 786)
112D	RSCNAM	<i>RSCNAM</i> (on page 781)
112E	PRDID	<i>PRDID</i> (on page 631)
113F	PRCCNVCD	<i>PRCCNVCD</i> (on page 621)
1144	VRSNAM	<i>VRSNAM</i> (on page 1059)
1147	SRVCLSNM	<i>SRVCLSNM</i> (on page 872)
1149	SVRCOD	<i>SVRCOD</i> (on page 911)
114A	SYNERRCD	<i>SYNERRCD</i> (on page 985)
114B	TEXT	<i>TEXT</i> (on page 1018)

Codepoint (Hex)	Term Name	Reference
1153	SRVDGN	<i>SRVDGN</i> (on page 873)
115A	SRVRLSLV	<i>SRVRLSLV</i> (on page 884)
115D	SPVNAM	<i>SPVNAM</i> (on page 834)
115E	EXTNAM	<i>EXTNAM</i> (on page 402)
116D	SRVNAM	<i>SRVNAM</i> (on page 880)
1184	LOGNAME	<i>LOGNAME</i> (on page 484)
1185	LOGTSTMP	<i>LOGTSTMP</i> (on page 485)
1186	FORGET	<i>FORGET</i> (on page 423)
1187	SYNCTYPE	<i>SYNCTYPE</i> (on page 984)
1196	SECMGRNM	<i>SECMGRNM</i> (on page 818)
119A	PKGDFTC	<i>PKGDFTC</i> (on page 589)
119B	DEPERRCD	<i>DEPERRCD</i> (on page 270)
119C	CCSIDSBC	<i>CCSIDSBC</i> (on page 154)
119D	CCSIDDBC	<i>CCSIDDBC</i> (on page 148)
119E	CCSIDMBC	<i>CCSIDMBC</i> (on page 149)
119F	RLSCONV	<i>RLSCONV</i> (on page 765)
11A0	USRID	<i>USRID</i> (on page 1044)
11A1	PASSWORD	<i>PASSWORD</i> (on page 578)
11A2	SECMEC	<i>SECMEC</i> (on page 811)
11A4	SECCHKCD	<i>SECCHKCD</i> (on page 806)
11AA	UOWID	<i>UOWID</i> (on page 1038)
11AC	UOWSTATE	<i>UOWSTATE</i> (on page 1040)
11B4	SVCERRNO	<i>SVCERRNO</i> (on page 910)
11DC	SECTKN	<i>SECTKN</i> (on page 820)
11DD	TCPHOST	<i>TCPHOST</i> (on page 999)
11DE	NEWPASSWORD	<i>NEWPASSWORD</i> (on page 531)
11E8	IPADDR	<i>IPADDR</i> (on page 450)
11E9	SNAADDR	<i>SNAADDR</i> (on page 827)
11EA	RSYNCTYP	<i>RSYNCTYP</i> (on page 790)
1210	MGRVLVLRM	<i>MGRVLVLRM</i> (on page 512)
1218	MGRDEPRM	<i>MGRDEPRM</i> (on page 505)
1219	SECCHKRM	<i>SECCHKRM</i> (on page 809)

*Term Names (Sorted by Codepoints)*

<b>Codepoint (Hex)</b>	<b>Term Name</b>	<b>Reference</b>
121C	CMDATHRM	<i>CMDATHRM</i> (on page 171)
1232	AGNPRMRM	<i>AGNPRMRM</i> (on page 65)
1233	RSCLMTRM	<i>RSCLMTRM</i> (on page 778)
1245	PRCCNVRM	<i>PRCCNVRM</i> (on page 625)
1248	SYNCCRD	<i>SYNCCRD</i> (on page 913)
124B	CMDCMPRM	<i>CMDCMPRM</i> (on page 175)
124C	SYNTAXRM	<i>SYNTAXRM</i> (on page 989)
1250	CMDNSPRM	<i>CMDNSPRM</i> (on page 176)
1251	PRMNSPRM	<i>PRMNSPRM</i> (on page 633)
1252	VALNSPRM	<i>VALNSPRM</i> (on page 1057)
1253	OBJNSPRM	<i>OBJNSPRM</i> (on page 542)
1254	CMDCHKRM	<i>CMDCHKRM</i> (on page 173)
125F	TRGNSPRM	<i>TRGNSPRM</i> (on page 1022)
126D	SYNCRRD	<i>SYNCRRD</i> (on page 981)
1403	AGENT	<i>AGENT</i> (on page 61)
1404	MGRLVLLS	<i>MGRLVLLS</i> (on page 508)
1408	CMNMGR	<i>CMNMGR</i> (on page 196)
1409	STGLMT	<i>STGLMT</i> (on page 885)
140D	DSSFMT	<i>DSSFMT</i> (on page 318)
1429	OBJDSS	<i>OBJDSS</i> (on page 536)
1436	RPYDSS	<i>RPYDSS</i> (on page 766)
1437	RPYMSG	<i>RPYMSG</i> (on page 770)
1438	RQSCRR	<i>RQSCRR</i> (on page 772)
1439	RQSDSS	<i>RQSDSS</i> (on page 774)
143C	SUPERVISOR	<i>SUPERVISOR</i> (on page 908)
1440	SECMGR	<i>SECMGR</i> (on page 814)
1442	MGRVLV	<i>MGRVLV</i> (on page 506)
1443	EXCSATRD	<i>EXCSATRD</i> (on page 369)
1444	CMNAPPC	<i>CMNAPPC</i> (on page 184)
1448	SERVER	<i>SERVER</i> (on page 824)
1450	DCTIND	<i>DCTIND</i> (on page 258)
1451	DCTINDEN	<i>DCTINDEN</i> (on page 259)

Codepoint (Hex)	Term Name	Reference
1452	MGRNAM	<i>MGRNAM</i> (on page 513)
1456	MANAGER	<i>MANAGER</i> (on page 491)
1458	DICTIONARY	<i>DICTIONARY</i> (on page 286)
146C	EXTDTA	<i>EXTDTA</i> (on page 397)
1473	MGRLVLN	<i>MGRLVLN</i> (on page 509)
1474	CMNTCPIP	<i>CMNTCPIP</i> (on page 214)
147A	FDODTA	<i>FDODTA</i> (on page 408)
147B	FDOEXT	<i>FDOEXT</i> (on page 410)
147C	CMNSYNCPT	<i>CMNSYNCPT</i> (on page 202)
147D	FDOOFF	<i>FDOOFF</i> (on page 412)
1480	FDOOBJ	<i>FDOOBJ</i> (on page 411)
14AC	ACCSECRD	<i>ACCSECRD</i> (on page 58)
14B1	UNORDERED	<i>UNORDERED</i> (on page 1036)
14C0	SYNCPTMGR	<i>SYNCPTMGR</i> (on page 939)
14C1	RSYNCMGR	<i>RSYNCMGR</i> (on page 787)
14CC	CCSIDMGR	<i>CCSIDMGR</i> (on page 150)
1801	XID	<i>XID</i> (on page 1109)
1802	XIDSHR	<i>XIDSHR</i> (on page 1112)
1803	BGNATMCHN	<i>BGNATMCHN</i> (on page 107)
1804	ENDATMCHN	<i>ENDATMCHN</i> (on page 333)
1805	SNDPKT	<i>SNDPKT</i> (on page 829)
1900	MONITOR	<i>MONITOR</i> (on page 521)
1901	ETIME	<i>ETIME</i> (on page 352)
1902	ENDCHNTYP	<i>ENDCHNTYP</i> (on page 341)
1903	XAFLAGS	<i>XAFLAGS</i> (on page 1061)
1904	XARETVAL	<i>XARETVAL</i> (on page 1107)
1905	PRPHRCLST	<i>PRPHRCLST</i> (on page 635)
1906	XIDCNT	<i>XIDCNT</i> (on page 1111)
1908	RESPKTSZ	<i>RESPKTSZ</i> (on page 763)
1909	ENCALG	<i>ENCALG</i> (on page 331)
190A	ENCKEYLEN	<i>ENCKEYLEN</i> (on page 332)
190B	SECTKNOVR	<i>SECTKNOVR</i> (on page 822)



*Term Names (Sorted by Codepoints)*

<b>Codepoint (Hex)</b>	<b>Term Name</b>	<b>Reference</b>
190D	PLGINID	<i>PLGINID</i> (on page 611)
190F	PLGINCNT	<i>PLGINCNT</i> (on page 610)
1910	PLGINLSE	<i>PLGINLSE</i> (on page 612)
1912	TCPPTHOST	<i>TCPPTHOST</i> (on page 1006)
191C	PLGINNM	<i>PLGINNM</i> (on page 614)
191E	PLGINLST	<i>PLGINLST</i> (on page 613)
1911	PLGINPPL	<i>PLGINPPL</i> (on page 619)
1C00	MONITORRD	<i>MONITORRD</i> (on page 523)
1C01	XAMGR	<i>XAMGR</i> (on page 1063)
1C02	KERSECPPL	<i>KERSECPPL</i> (on page 470)
1C03	MGRLVLOVR	<i>MGRLVLOVR</i> (on page 511)
1C04	PKTOBJ	<i>PKTOBJ</i> (on page 608)
2001	ACCRDB	<i>ACCRDB</i> (on page 42)
2002	BGNBND	<i>BGNBND</i> (on page 110)
2003	INTRDBRQS	<i>INTRDBRQS</i> (on page 445)
2004	BNDSQLSTT	<i>BNDSQLSTT</i> (on page 136)
2005	CLSQRY	<i>CLSQRY</i> (on page 165)
2006	CNTQRY	<i>CNTQRY</i> (on page 222)
2007	DRPPKG	<i>DRPPKG</i> (on page 293)
2008	DSCSQLSTT	<i>DSCSQLSTT</i> (on page 304)
2009	ENDBND	<i>ENDBND</i> (on page 336)
200A	EXCSQLIMM	<i>EXCSQLIMM</i> (on page 371)
200B	EXCSQLSTT	<i>EXCSQLSTT</i> (on page 381)
200C	OPNQRY	<i>OPNQRY</i> (on page 555)
200D	PRPSQLSTT	<i>PRPSQLSTT</i> (on page 636)
200E	RDBCMM	<i>RDBCMM</i> (on page 728)
200F	RDBRLLBCK	<i>RDBRLLBCK</i> (on page 745)
2010	REBIND	<i>REBIND</i> (on page 753)
2012	DSCRDBTBL	<i>DSCRDBTBL</i> (on page 300)
2014	EXCSQLSET	<i>EXCSQLSET</i> (on page 377)
2101	DSCERRCD	<i>DSCERRCD</i> (on page 296)
2102	QRYPRCTYP	<i>QRYPRCTYP</i> (on page 692)

Codepoint (Hex)	Term Name	Reference
2103	RDBINTTKN	<i>RDBINTTKN</i> (on page 733)
2104	PRDDTA	<i>PRDDTA</i> (on page 630)
2105	RDBCMTOK	<i>RDBCMTOK</i> (on page 731)
2106	DECPRC	<i>DECPRC</i> (on page 266)
2107	CMDSRCID	<i>CMDSRCID</i> (on page 178)
2108	RDBCOLID	<i>RDBCOLID</i> (on page 732)
2109	PKGID	<i>PKGID</i> (on page 591)
210A	PKGNAM	<i>PKGNAM</i> (on page 594)
210B	DUPQRYOK	<i>DUPQRYOK</i> (on page 322)
210C	PKGSN	<i>PKGSN</i> (on page 606)
210D	PKGCNSTKN	<i>PKGCNSTKN</i> (on page 588)
210E	RTNSETSTT	<i>RTNSETSTT</i> (on page 794)
210F	RDBACCCL	<i>RDBACCCL</i> (on page 723)
2110	RDBNAM	<i>RDBNAM</i> (on page 736)
2111	OUTEXP	<i>OUTEXP</i> (on page 571)
2112	PKGNAMCT	<i>PKGNAMCT</i> (on page 598)
2113	PKGNAMCSN	<i>PKGNAMCSN</i> (on page 596)
2114	QRYBLKSZ	<i>QRYBLKSZ</i> (on page 678)
2115	UOWDSP	<i>UOWDSP</i> (on page 1037)
2116	RTNSQLDA	<i>RTNSQLDA</i> (on page 795)
2117	SQLSTTNBR	<i>SQLSTTNBR</i> (on page 870)
2118	FDODSCOFF	<i>FDODSCOFF</i> (on page 407)
2119	FDODTAOFF	<i>FDODTAOFF</i> (on page 409)
211A	RDBALWUPD	<i>RDBALWUPD</i> (on page 726)
211B	BNDCHKEXS	<i>BNDCHKEXS</i> (on page 125)
211C	PKGRPLOPT	<i>PKGRPLOPT</i> (on page 604)
211D	BNDCRTCTL	<i>BNDCRTCTL</i> (on page 127)
211E	PKGATHOPT	<i>PKGATHOPT</i> (on page 580)
211F	SQLCSRHLD	<i>SQLCSRHLD</i> (on page 858)
2120	STTSTRDEL	<i>STTSTRDEL</i> (on page 899)
2121	STTDECDEL	<i>STTDECDEL</i> (on page 897)
2122	STTDATFMT	<i>STTDATFMT</i> (on page 894)

*Term Names (Sorted by Codepoints)*

<b>Codepoint (Hex)</b>	<b>Term Name</b>	<b>Reference</b>
2123	STTTIMFMT	<i>STTTIMFMT</i> (on page 900)
2124	PKGISOLVL	<i>PKGISOLVL</i> (on page 592)
2125	PKGDFTCST	<i>PKGDFTCST</i> (on page 590)
2126	BNDSTTASM	<i>BNDSTTASM</i> (on page 141)
2127	MAXSCTNBR	<i>MAXSCTNBR</i> (on page 498)
2128	DFTRDBCOL	<i>DFTRDBCOL</i> (on page 274)
2129	RDBRLSOPT	<i>RDBRLSOPT</i> (on page 750)
212A	FDOTRPOFF	<i>FDOTRPOFF</i> (on page 414)
212B	FDOPRMOFF	<i>FDOPRMOFF</i> (on page 413)
212D	PKGRPLVRS	<i>PKGRPLVRS</i> (on page 605)
212F	DGRIOPRL	<i>DGRIOPRL</i> (on page 279)
2130	BNDEXPOPT	<i>BNDEXPOPT</i> (on page 129)
2131	PKGOWNID	<i>PKGOWNID</i> (on page 600)
2132	QRYBLKCTL	<i>QRYBLKCTL</i> (on page 672)
2133	QRYBLKTYP	<i>QRYBLKTYP</i> (on page 679)
2135	CRRTKN	<i>CRRTKN</i> (on page 246)
2138	PRCNAM	<i>PRCNAM</i> (on page 627)
2139	PKGSNLST	<i>PKGSNLST</i> (on page 607)
213A	NBRROW	<i>NBRROW</i> (on page 530)
213D	QRYROWNBR	<i>QRYROWNBR</i> (on page 694)
213F	PKGATHRUL	<i>PKGATHRUL</i> (on page 581)
2140	MAXRSLCNT	<i>MAXRSLCNT</i> (on page 497)
2141	MAXBLKEXT	<i>MAXBLKEXT</i> (on page 494)
2142	RSLSETFLG	<i>RSLSETFLG</i> (on page 783)
2143	CMMTYP	<i>CMMTYP</i> (on page 183)
2144	BNDOPTNM	<i>BNDOPTNM</i> (on page 134)
2145	BNDOPTVL	<i>BNDOPTVL</i> (on page 135)
2146	TYPESQLDA	<i>TYPESQLDA</i> (on page 1034)
2147	OUTOVROPT	<i>OUTOVROPT</i> (on page 576)
2148	RTNEXTDTA	<i>RTNEXTDTA</i> (on page 792)
2149	QRYATTSCR	<i>QRYATTSCR</i> (on page 665)
214A	QRYATTSET	<i>QRYATTSET</i> (on page 666)

Codepoint (Hex)	Term Name	Reference
2150	QRYATTUPD	<i>QRYATTUPD</i> (on page 669)
2152	QRYSCRORN	<i>QRYSCRORN</i> (on page 710)
2153	QRYROWSNS	<i>QRYROWSNS</i> (on page 701)
2154	QRYBLKRST	<i>QRYBLKRST</i> (on page 677)
2155	QRYRTNDTA	<i>QRYRTNDTA</i> (on page 702)
2156	QRYROWSET	<i>QRYROWSET</i> (on page 697)
2157	QRYATTSNS	<i>QRYATTSNS</i> (on page 667)
2159	ATMIND	<i>ATMIND</i> (on page 104)
215B	QRYINSID	<i>QRYINSID</i> (on page 688)
215C	BUFINSIND	<i>BUFINSIND</i> (on page 143)
215D	QRYCLSIMP	<i>QRYCLSIMP</i> (on page 680)
215E	QRYCLSRLS	<i>QRYCLSRLS</i> (on page 682)
215F	QRYBLKFCT	<i>QRYBLKFCT</i> (on page 675)
2160	DIAGLVL	<i>DIAGLVL</i> (on page 284)
2161	ARMCORR	<i>ARMCORR</i> (on page 100)
2167	PRPSTTKP	<i>PRPSTTKP</i> (on page 642)
2201	ACCRDBRM	<i>ACCRDBRM</i> (on page 48)
2202	QRYNOPRM	<i>QRYNOPRM</i> (on page 690)
2203	RDBATHRM	<i>RDBATHRM</i> (on page 727)
2204	RDBNACRM	<i>RDBNACRM</i> (on page 734)
2205	OPNQRYRM	<i>OPNQRYRM</i> (on page 566)
2206	PKGBNARM	<i>PKGBNARM</i> (on page 585)
2207	RDBACCRM	<i>RDBACCRM</i> (on page 724)
2208	BGNBNDRM	<i>BGNBNDRM</i> (on page 117)
2209	PKGBPARM	<i>PKGBPARM</i> (on page 586)
220A	DSCINVRM	<i>DSCINVRM</i> (on page 298)
220B	ENDQRYRM	<i>ENDQRYRM</i> (on page 342)
220C	ENDUOWRM	<i>ENDUOWRM</i> (on page 343)
220D	ABNUOWRM	<i>ABNUOWRM</i> (on page 39)
220E	DTAMCHRM	<i>DTAMCHRM</i> (on page 320)
220F	QRYPOPRM	<i>QRYPOPRM</i> (on page 691)
2210	INTTKNRM	<i>INTTKNRM</i> (on page 448)

*Term Names (Sorted by Codepoints)*

<b>Codepoint (Hex)</b>	<b>Term Name</b>	<b>Reference</b>
2211	RDBNFNRM	<i>RDBNFNRM</i> (on page 739)
2212	OPNQFLRM	<i>OPNQFLRM</i> (on page 554)
2213	SQLERRRM	<i>SQLERRRM</i> (on page 864)
2218	RDBUPDRM	<i>RDBUPDRM</i> (on page 751)
2219	RSLSETRM	<i>RSLSETRM</i> (on page 785)
221A	RDBAFLRM	<i>RDBAFLRM</i> (on page 725)
221D	CMDVLTRM	<i>CMDVLTRM</i> (on page 181)
2225	CMMRQSRM	<i>CMMRQSRM</i> (on page 182)
2400	DFTDATFMT	<i>DFTDATFMT</i> (on page 272)
2401	DFTTIMFMT	<i>DFTTIMFMT</i> (on page 275)
2402	DMYBLKDATFMT	<i>DMYBLKDATFMT</i> (on page 288)
2403	DMYCMADATFMT	<i>DMYCMADATFMT</i> (on page 289)
2404	DMYHPNDATFMT	<i>DMYHPNDATFMT</i> (on page 290)
2405	BNDOPT	<i>BNDOPT</i> (on page 133)
2406	DMYPRDDATFMT	<i>DMYPRDDATFMT</i> (on page 291)
2407	SQLAM	<i>SQLAM</i> (on page 847)
2408	SQLCARD	<i>SQLCARD</i> (on page 855)
2409	DMYSLHDATFMT	<i>DMYSLHDATFMT</i> (on page 292)
240A	EXPYES	<i>EXPYES</i> (on page 396)
240B	SQLCINRD	<i>SQLCINRD</i> (on page 857)
240C	HMSBLKTIMFMT	<i>HMSBLKTIMFMT</i> (on page 431)
240D	HMSCMATIMFMT	<i>HMSCMATIMFMT</i> (on page 433)
240E	SQLRSLRD	<i>SQLRSLRD</i> (on page 867)
240F	RDB	<i>RDB</i> (on page 718)
2410	FRCFIXROW	<i>FRCFIXROW</i> (on page 424)
2411	SQLDARD	<i>SQLDARD</i> (on page 859)
2412	SQLDTA	<i>SQLDTA</i> (on page 860)
2413	SQLDTARD	<i>SQLDTARD</i> (on page 862)
2414	SQLSTT	<i>SQLSTT</i> (on page 868)
2415	OUTOVR	<i>OUTOVR</i> (on page 572)
2417	LMTBLKPRC	<i>LMTBLKPRC</i> (on page 475)
2418	FIXROWPRC	<i>FIXROWPRC</i> (on page 417)

Codepoint (Hex)	Term Name	Reference
2419	SQLSTTVRB	<i>SQLSTTVRB</i> (on page 871)
241A	QRYDSC	<i>QRYDSC</i> (on page 685)
241B	QRYDTA	<i>QRYDTA</i> (on page 686)
241C	BNDEXSRQR	<i>BNDEXSRQR</i> (on page 131)
241D	BNDEXSOPT	<i>BNDEXSOPT</i> (on page 130)
241E	DFTPKG	<i>DFTPKG</i> (on page 273)
241F	PKGRPLALW	<i>PKGRPLALW</i> (on page 602)
2420	PKGRPLNA	<i>PKGRPLNA</i> (on page 603)
2421	BNDCHKONL	<i>BNDCHKONL</i> (on page 126)
2422	BNDNERALW	<i>BNDNERALW</i> (on page 132)
2423	BNDERRALW	<i>BNDERRALW</i> (on page 128)
2424	PKGATHRVK	<i>PKGATHRVK</i> (on page 584)
2425	PKGATHKP	<i>PKGATHKP</i> (on page 579)
2426	STRDELAP	<i>STRDELAP</i> (on page 886)
2427	STRDELDQ	<i>STRDELDQ</i> (on page 887)
2428	HMSPRDTIMFMT	<i>HMSPRDTIMFMT</i> (on page 434)
2429	ISODATFMT	<i>ISODATFMT</i> (on page 451)
242A	USADATFMT	<i>USADATFMT</i> (on page 1041)
242B	EURDATFMT	<i>EURDATFMT</i> (on page 353)
242C	JISDATFMT	<i>JISDATFMT</i> (on page 458)
242D	JULBLKDATFMT	<i>JULBLKDATFMT</i> (on page 460)
242E	ISOTIMFMT	<i>ISOTIMFMT</i> (on page 457)
242F	USATIMFMT	<i>USATIMFMT</i> (on page 1042)
2430	EURTIMFMT	<i>EURTIMFMT</i> (on page 354)
2431	JISTIMFMT	<i>JISTIMFMT</i> (on page 459)
2432	CSTSYSDFT	<i>CSTSYSDFT</i> (on page 250)
2433	CSTBITS	<i>CSTBITS</i> (on page 247)
2434	CSTSBCS	<i>CSTSBCS</i> (on page 249)
2435	CSTMBCS	<i>CSTMBCS</i> (on page 248)
2436	STTSCCLS	<i>STTSCCLS</i> (on page 898)
2437	STTASMEUI	<i>STTASMEUI</i> (on page 893)
2438	RDBRLSCMM	<i>RDBRLSCMM</i> (on page 748)

*Term Names (Sorted by Codepoints)*

<b>Codepoint (Hex)</b>	<b>Term Name</b>	<b>Reference</b>
2439	RDBRLSCNV	<i>RDBRLSCNV</i> (on page 749)
243A	EXPNON	<i>EXPNON</i> (on page 394)
243B	EXPALL	<i>EXPALL</i> (on page 393)
243C	DECDELPRD	<i>DECDELPRD</i> (on page 265)
243D	DECDELCPA	<i>DECDELCPA</i> (on page 264)
243E	SQLOBJNAM	<i>SQLOBJNAM</i> (on page 866)
243F	JULCMADATFMT	<i>JULCMADATFMT</i> (on page 461)
2440	JULHPNDATFMT	<i>JULHPNDATFMT</i> (on page 462)
2441	ISOLVLCHG	<i>ISOLVLCHG</i> (on page 453)
2442	ISOLVLCS	<i>ISOLVLCS</i> (on page 454)
2443	ISOLVLALL	<i>ISOLVLALL</i> (on page 452)
2444	ISOLVLR	<i>ISOLVLR</i> (on page 456)
2445	ISOLVLNC	<i>ISOLVLNC</i> (on page 455)
2446	JULPRDDATFMT	<i>JULPRDDATFMT</i> (on page 463)
2447	JULSLHDATFMT	<i>JULSLHDATFMT</i> (on page 464)
2448	LOCDFMT	<i>LOCDFMT</i> (on page 482)
2449	LOCTIMFMT	<i>LOCTIMFMT</i> (on page 483)
244A	MDYBLKDFMT	<i>MDYBLKDFMT</i> (on page 500)
244B	MDYCMADATFMT	<i>MDYCMADATFMT</i> (on page 501)
244C	SRVLCNT	<i>SRVLCNT</i> (on page 875)
244D	SRVLSRV	<i>SRVLSRV</i> (on page 876)
244E	SRVLST	<i>SRVLST</i> (on page 878)
244F	SRVPTY	<i>SRVPTY</i> (on page 883)
2450	SQLATTR	<i>SQLATTR</i> (on page 854)
2451	MDYHPNDATFMT	<i>MDYHPNDATFMT</i> (on page 502)
2452	MDYPRDDATFMT	<i>MDYPRDDATFMT</i> (on page 503)
2453	MDYSLHDATFMT	<i>MDYSLHDATFMT</i> (on page 504)
2454	YMDBLKDFMT	<i>YMDBLKDFMT</i> (on page 1113)
2455	YMDCMADATFMT	<i>YMDCMADATFMT</i> (on page 1114)
2456	YMDHPNDATFMT	<i>YMDHPNDATFMT</i> (on page 1115)
2457	YMDPRDDATFMT	<i>YMDPRDDATFMT</i> (on page 1116)
2458	YMDSLHDATFMT	<i>YMDSLHDATFMT</i> (on page 1117)
2459	EXPLOPT	<i>EXPLOPT</i> (on page 395)





## DDM Codepoint Scheme

### Main DRDA Dictionaries

The main DRDA dictionaries are as follows:

X'0xxx'	QDDPRMD Primary Dictionary for DDM primitives.
X'1xxx'	QDDBASD Base Dictionary for General Command Flows.
X'2xxx'	QDDRDBD RDB Dictionary for RDB Command Flows.

X'Cxxx' through X'Fxxx' are reserved for private extensions.

All unused X'1001' through X'17FF' codepoints are to be avoided due to conflicts with old file codepoints.

### New QDDBASD Assignments

New QDDBASD assignments should be made in the following ranges:

X'18xx'	For commands.
X'19xx'	For parameters.
X'1Axx'	For reply messages.
X'1Cxx'	For other classes of objects.

### New QDDRDB Assignments

New QDDRDB assignments should be made in the following ranges:

X'20xx'	For commands.
X'21xx'	For parameters.
X'22xx'	For reply messages.
X'24xx'	For other classes of objects.

### Additional Ranges for QDDRDB

Additional ranges for QDDRDB are predefined for when there are no available values in the existing range:

X'28xx'	For commands.
X'29xx'	For parameters.
X'2Axx'	For reply messages.
X'2Cxx'	For other classes of objects.

### Coding for Classes of Objects

To summarize, the last three bits of the first code-point byte should be:

000	For commands.
001	For parameters.
010	For reply messages.
100	For other classes of objects.



# Index

ABBREVIATIONS.....	30	BUFINSIND.....	143
ABNUOWRM.....	39	BYTDR.....	145-146
ACCDMG.....	41	BYTSTRDR.....	146
ACCRDB.....	42	CCSIDDBC.....	148
ACCRDBRM.....	48	CCSIDMBC.....	149
ACCSEC.....	52	CCSIDMGR.....	150
ACCSECRD.....	58	CCSIDSBC.....	154
agent.....	15	CHRDR.....	155
AGENT.....	61	CHRSTRDR.....	156
AGNCMDPR.....	64	CLASS.....	158
AGNPRMRM.....	65	CLSQR.....	165
AGNRPYPR.....	67	CMDATHRM.....	171
APPCMNFL.....	68	CMDCHKRM.....	173
APPCMNI.....	72	CMDCMPRM.....	175
APPCMNT.....	76	CMDNSPRM.....	176
APPSRCCD.....	79	CMDSRCID.....	178
APPSRCCR.....	86	CMDTRG.....	179
APPSRCER.....	91	CMDVLTRM.....	181
APPTRGER.....	95	CMMRQSRM.....	182
ARMCORR.....	100	CMMTYP.....	183
ARRAY.....	101	CMNAPPC.....	184
ASSOCIATION.....	103	CMNLYR.....	194
ATMIND.....	104	CMNMGR.....	196
ATTLST.....	105	CMNOVR.....	201
base data object.....	399	CMNSYNCPT.....	202
BGNATMCHN.....	107	CMNTCPIP.....	214
BGNBND.....	110	CNNTKN.....	220
BGNBNDRM.....	117	CNSVAL.....	221
BIN.....	118	CNTQRY.....	222
BINDR.....	120	CODPNT.....	233
BITDR.....	123	CODPNTDR.....	235
BITSTRDR.....	124	COLLECTION.....	238
BNDCHKEXS.....	125	COMMAND.....	240
BNDCHKONL.....	126	communications manager.....	15
BNDCRTCTL.....	127	CONCEPTS.....	243
BNDERRALW.....	128	Connection.....	1003
BNDEXPOPT.....	129	CONSTANT.....	244
BNDEXSOPT.....	130	CRRTKN.....	246
BNDEXSRQR.....	131	CSTBITS.....	247
BNDNERALW.....	132	CSTMBCS.....	248
BNDOPT.....	133	CSTSBCS.....	249
BNDOPTNM.....	134	CSTSYSDFT.....	250
BNDOPTVL.....	135	data.....	14
BNDSQLSTT.....	136	DATA.....	251
BNDSTTASM.....	141	data conversion.....	17
BOOLEAN.....	142	Data Encryption Algorithm.....	282

Datagram.....	1003	DUPQRYOK.....	322
DCESEC.....	252	EDTASECOVR.....	323
DCESECOVR.....	253	ELMCLS.....	330
DCESECTKN.....	257	ENCALG.....	331
DCTIND.....	258	ENCKEYLEN.....	332
DCTINDEN.....	259	ENDATMCHN.....	333
DDM.....	260	ENDBND.....	336
DDM command objects in DRDA.....	5	ENDCHNTYP.....	341
DDM concepts for DRDA implementation.....	4	ENDQRYRM.....	342
DDM overview concepts.....	3	ENDUOWRM.....	343
DDM reader guide.....	3	ENUCLS.....	345
DDMID.....	261	ENULEN.....	346
DDMOBJ.....	262	ENUVAL.....	347
DEA-1.....	282	ERROR.....	350
DECDELCA.....	264	ETIME.....	352
DECDELPRD.....	265	EURDATFMT.....	353
DECPRC.....	266	EURTIMFMT.....	354
DEFINITION.....	267	EUSRIDDTA.....	355
DEFLST.....	268	EUSRIDNWPWD.....	357
DEPERECD.....	270	EUSRIDPWD.....	358
DES Standard.....	282	EUSRNPWDDTA.....	359
descriptor area.....	305, 384, 557, 637	EUSRPWDDTA.....	361
Destination Address.....	1003	exact blocking.....	670
DFTDATFMT.....	272	EXCSAT.....	363
DFTPKG.....	273	EXCSATRD.....	369
DFTRDBCOL.....	274	EXCSQLIMM.....	371
DFTTIMFMT.....	275	EXCSQLSET.....	377
DFTVAL.....	276	EXCSQLSTT.....	381
DGRIOPRL.....	279	EXPALL.....	393
DHENC.....	280	EXPNON.....	394
DIAGLVL.....	284	EXPLOPT.....	395
dictionary.....	16	EXPYES.....	396
DICTIONARY.....	286	EXTDTA.....	397
Directory.....	1003	EXTDTAOVR.....	399
distributed processing model.....	12	EXTENSIONS.....	400
DMYBLKDATFMT.....	288	EXTNAM.....	402
DMYCMADATFMT.....	289	FALSE.....	403
DMYHPNDATFMT.....	290	FDOCA.....	404
DMYPRDDATFMT.....	291	FDODSC.....	406
DMYSLHDATFMT.....	292	FDODSCOFF.....	407
DNS.....	878	FDODTA.....	408
DRPPKG.....	293	FDODTAOFF.....	409
DSCERRCD.....	296	FDOEXT.....	410
DSCINVRM.....	298	FDOOBJ.....	411
DSCRDBTBL.....	300	FDOOFF.....	412
DSCSQLSTT.....	304	FDOPRMOFF.....	413
DSS.....	308	FDOTRPOFF.....	414
DSS encryption.....	315	FIELD.....	415
DSSFMT.....	318	FIXROWPRC.....	417
DTAMCHRM.....	320	flexible blocking.....	670
DTAOVR.....	321	FORGET.....	423

## Index

Fragment .....	1003	MAXBLKEXT .....	494
FRCFIXROW .....	424	MAXLEN.....	495
FTP.....	1003	MAXRSLCNT.....	497
Header .....	1003	MAXSCTNBR.....	498
HELP .....	425	MAXVAL.....	499
HEXDR .....	428	MDYBLKDATFMT .....	500
HEXSTRDR.....	429	MDYCMADATFMT .....	501
HMSBLKTIMFMT .....	431	MDYHPNDATFMT .....	502
HMSCLNTIMFMT .....	432	MDYPRDDATFMT .....	503
HMSCMATIMFMT .....	433	MDYSLHDATFMT .....	504
HMSPRDTIMFMT .....	434	MGRDEPRM .....	505
Host .....	1004	MGRVLV.....	506
IGNORABLE .....	435	MGRVLVLS.....	508
INFO.....	436	MGRVLVN .....	509
INHERITANCE.....	437	MGRVLVLOVR .....	511
INHERITED.....	442	MGRVLVLRM.....	512
INSTANCE_OF.....	444	MGRNAM.....	513
Internet Address.....	1004	MGROVR .....	514
Internet Datagram.....	1004	MINLEN .....	517
Internet Fragment.....	1004	MINLVL.....	518
INTRDBRQS .....	445	MINVAL.....	519
INTTKNRM.....	448	MONITOR .....	521
IP .....	1004	MONITORRD .....	523
IPADDR.....	450	MTLEXC.....	524
ISODATFMT .....	451	MTLINC .....	525
ISOLVLALL .....	452	NAMDR.....	526
ISOLVLCHG.....	453	NAME.....	527
ISOLVLCS .....	454	NAMSYMDR.....	528
ISOLVLNC .....	455	NBRROW .....	530
ISOLVLR .....	456	NEWPASSWORD .....	531
ISOTIMFMT.....	457	NIL.....	532
JISDATFMT .....	458	NOTE .....	533
JISTIMFMT .....	459	NUMBER.....	534
JULBLKDATFMT .....	460	NWPWDSEC.....	535
JULCMADATFMT .....	461	OBJDSS .....	536
JULHPNDATFMT .....	462	object .....	14
JULPRDDATFMT .....	463	OBJECT .....	540
JULSLHDATFMT .....	464	OBJNSPRM .....	542
KERSEC .....	465	OBJOVR .....	544
KERSECOVR .....	466	Octet.....	1004
KERSECPPL.....	470	OOPOVR.....	547
KERSECTKN .....	471	OPNQFLRM.....	554
LENGTH .....	472	OPNQRY .....	555
LMTBLKPRC .....	475	OPNQRYRM .....	566
LOCDATFMT .....	482	OPTIONAL.....	568
LOCTIMFMT.....	483	ORDCOL.....	569
LOGNAME .....	484	OSFDCE.....	570
LOGTSTMP .....	485	OUTEXP .....	571
LVLCMP .....	486	OUTOVR.....	572
manager.....	14	OUTOVRANY .....	574
MANAGER.....	491	OUTOVRFRS.....	575

OUTOVROPT.....	576	PWDSEC.....	649
OWNER.....	577	PWDSSB .....	650
Packet.....	1004	QDDBASD .....	651
PASSWORD.....	578	QDDPRMD .....	654
password substitute.....	648	QDDRDBD.....	657
PKGATHKP .....	579	QDDTTRD .....	662
PKGATHOPT .....	580	QLFATT .....	664
PKGATHRUL.....	581	QRYATTSCR.....	665
PKGATHRVK.....	584	QRYATTSET .....	666
PKGBNARM.....	585	QRYATTSNS.....	667
PKGBPARM.....	586	QRYATTUPD.....	669
PKGCNSTKN.....	588	QRYBLK .....	670
PKGDFTCC .....	589	QRYBLKCTL .....	672
PKGDFTCST.....	590	QRYBLKEXA.....	674
PKGID.....	591	QRYBLKFCT .....	675
PKGISOLVL.....	592	QRYBLKFLX.....	676
PKGNAME.....	594	QRYBLKRST.....	677
PKGNAMECSN .....	596	QRYBLKSZ .....	678
PKGNAMECT .....	598	QRYBLKTYP .....	679
PKGOWNID.....	600	QRYCLSIMP.....	680
PKGRPLALW.....	602	QRYCLSRLS.....	682
PKGRPLNA .....	603	QRYDEL.....	684, 687
PKGRPLOPT .....	604	QRYDSC.....	685
PKGRPLVRS.....	605	QRYDTA.....	686
PKGSN.....	606	QRYINS .....	687
PKGSNLST .....	607	QRYINSID.....	688
PKTOBJ.....	608	QRYNOPRM .....	690
PLGIN .....	609	QRYPOPRM .....	691
PLGINCNT.....	610	QRYPRCTYP .....	692
PLGINID .....	611	QRYRDO.....	693
PLGINLSE.....	612	QRYROWNBR .....	694
PLGINLST.....	613	QRYROWSET .....	697
PLGINNM.....	614	QRYROWSNS .....	701
PLGINOVR.....	615	QRYRTNDDTA.....	702
PLGINPPL.....	619	QRYSCRABS .....	703
PLGINSECTKN .....	620	QRYSCRAFT .....	704
Port .....	1004	QRYSCRBEP .....	705
PRCCNVCD.....	621	QRYSCRCUR .....	706
PRCCNVRM.....	625	QRYSCRFST .....	707
PRCNAM .....	627	QRYSCRLST.....	708
PRCOVR.....	628	QRYSCRNXT .....	709
PRDDTA.....	630	QRYSCRORN.....	710
PRDID .....	631	QRYSCRPRI.....	712
PRMDMG.....	632	QRYSCRREL .....	713
PRMNSPRM.....	633	QRYSNSDYN.....	714
Process .....	1004	QRYSNSSTC.....	715
PRPHRCLST.....	635	QRYUNK.....	716
PRPSQLSTT .....	636	QRYUPD .....	717
PRPSTTKP .....	642	RDB .....	12, 718
PWDENC .....	644	RDB manager.....	16
PWDSBS .....	646	RDBACCCL.....	723

## Index

RDBACCRM.....	724	SECMGRNM.....	818
RDBAFLRM.....	725	SECOVR.....	819
RDBALWUPD.....	726	SECTKN.....	820
RDBATHRM.....	727	SECTKNOVR.....	822
RDBCMM.....	728	security manager.....	16
RDBCMTOK.....	731	Segment.....	1004
RDBCOLID.....	732	server.....	14
RDBINTTKN.....	733	SERVER.....	824
RDBNACRM.....	734	SESDMG.....	825
RDBNAM.....	736	SEVERE.....	826
RDBNFNRM.....	739	SNAADDR.....	827
RDBOVR.....	740	SNASECOVR.....	828
RDBRLBCK.....	745	SNDPKT.....	829
RDBRLSCMM.....	748	Socket.....	1004
RDBRLSCNV.....	749	Source Address.....	1004
RDBRISOPT.....	750	SPCVAL.....	831
RDBUPDRM.....	751	SPRCLS.....	832
REBIND.....	753	SPVNAM.....	834
relational database model.....	11	SQL.....	835
REPEATABLE.....	758	SQL application manager.....	17
reply objects and messages.....	6	SQLAM.....	13, 847
REQUESTER.....	760	SQLATTR.....	854
REQUIRED.....	761	SQLCARD.....	855
RESERVED.....	762	SQLCINRD.....	857
RESPKTSZ.....	763	SQLCSRHLD.....	858
RESYNOVR.....	764	SQLDARD.....	859
RLSCONV.....	765	SQLDTA.....	860
RPYDSS.....	766	SQLDTARD.....	862
RPYMSG.....	770	SQLERRRM.....	864
RQSCRR.....	772	SQLOBINAM.....	866
RQSDSS.....	774	SQLRSLRD.....	867
RSCLMTRM.....	778	SQLSTT.....	868
RSCNAM.....	781	SQLSTTNBR.....	870
RSCTYP.....	782	SQLSTTVRB.....	871
RSLSETFLG.....	783	SRVCLSNM.....	872
RSLSETRM.....	785	SRVDGN.....	873
RSNCOD.....	786	SRVLCNT.....	875
RSYNMGR.....	787	SRVLSRV.....	876
RSYNCTYP.....	790	SRVLST.....	878
RTNEXTALL.....	791	SRVNAM.....	880
RTNEXTDTA.....	792	SRVOVR.....	881
RTNEXTROW.....	793	SRVPRTY.....	883
RTNSETSTT.....	794	SRVRLSLV.....	884
RTNSQLDA.....	795	STGLMT.....	885
SCALAR.....	796	STRDELAP.....	886
SCLDTALEN.....	799	STRDELDQ.....	887
SECCHK.....	800	STRING.....	888
SECCHKCD.....	806	STRLYR.....	890
SECCHKRM.....	809	STTASMEUI.....	893
SECMEC.....	811	STTDATFMT.....	894
SECMGR.....	814	STTDECDEL.....	897

STTSCCCLS.....	898	UOWSTATE.....	1040
STTSTRDEL.....	899	USADATFMT.....	1041
STTTIMFMT.....	900	USATIMFMT.....	1042
SUBSETS.....	902	USRENCPWD.....	1043
supervisor.....	16	USRID.....	1044
SUPERVISOR.....	908	USRIDNWPWD.....	1045
SVCERRNO.....	910	USRIDONL.....	1046
SVRCOD.....	911	USRIDPWD.....	1047
sync point manager.....	17	USRIDSEC.....	1048
SYNCCRD.....	913	USRSBSPWD.....	1049
SYNCCTL.....	915	USRSECOVR.....	1050
SYNCLOG.....	922	USRSSBPWD.....	1056
SYNCMNBK.....	924	VALNSPRM.....	1057
SYNCMNCM.....	926	VRSNAM.....	1059
SYNCMNFL.....	928	WARNING.....	1060
SYNCMNI.....	931	Well-known Port.....	1004
SYNCMNT.....	935	XA manager.....	18
SYNCPTMGR.....	939	XAFLAGS.....	1061
SYNCPTOV.....	944	XAMGR.....	1063
SYNCRRD.....	981	XAMGROV.....	1066
SYNCRSY.....	982	XARETVAL.....	1107
SYNCTYPE.....	984	XID.....	1109
SYNERRCD.....	985	XIDCNT.....	1111
SYNTAXRM.....	989	XIDSHR.....	1112
TASK.....	991	YMDBLKDATFMT.....	1113
TCB.....	1004	YMDCMADATFMT.....	1114
TCP.....	1004	YMDHPNDATFMT.....	1115
TCPCMNFL.....	992	YMDPRDDATFMT.....	1116
TCPCMNI.....	994	YMDSLHDATFMT.....	1117
TCPCMNT.....	997		
TCPHOST.....	999		
TCPIPOVR.....	1000		
TCPPORHOST.....	1006		
TCPSRCCD.....	1007		
TCPSRCCR.....	1010		
TCPSRCER.....	1013		
TCPTRGER.....	1015		
TEXT.....	1018		
TIMEOUT.....	1019		
TITLE.....	1020		
TRGNSPRM.....	1022		
TRUE.....	1024		
TYPDEF.....	1025		
TYPDEFNAM.....	1027		
TYPDEFOVR.....	1030		
TYPFMLNM.....	1033		
TYPSQLDA.....	1034		
UDP.....	1004		
UNORDERED.....	1036		
UOWDSP.....	1037		
UOWID.....	1038		